WO0195107

Publication Title:

DISTRIBUTED COMPUTER SYSTEM USING A GRAPHICAL USER INTERFACE TOOLKIT

Abstract:

Abstract of WO0195107

A distributed computer system having a server and remote client for executing an application on the server. A remote-capable user interface toolkit resides on the server and has remote-capable components that correspond to components of a user interface toolkit which resides on the remote client. The remote-capable components are substantially the same as corresponding components of the user interface toolkit, and interact with the application according to the same application programming interface. However, when invoked by the application, the remote-capable components issue a message to the component on the remote client to perform the corresponding function on the client. A network communication protocol of sending messages between the remote-capable user interface toolkit on the server and the user interface toolkit on the client is thereby generated. The remote-capable components may be created by a code-generating routine which reads in the component of the user interface toolkit, copies the code of the component, and substitutes a portion of the code relevant to performing the function of the component with a portion of code that iss f14 ues a remote message to a component on a remote client to perform the same function.

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
13 December 2001 (13.12.2001)

PCT

(10) International Publication Number
WO 01/95107 A2

(51) International Patent Classification[7]: G06F 9/54

(21) International Application Number: PCT/US01/18703

(22) International Filing Date: 11 June 2001 (11.06.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/210,643 9 June 2000 (09.06.2000) US
60/277,498 21 March 2001 (21.03.2001) US

(71) Applicant: **THE TRUSTEES OF COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK** [US/US]; 116th Street and Broadway, New York, NY 10027 (US).

(72) Inventors: **LOK, Simon**; 164-38 85th Street, Howard Beach, NY 11414 (US). **FEINER, Steven, K.**; 90 Morningside Drive, New York, NY 10027 (US).

(74) Agents: **TANG, Henry** et al.; Baker & Botts LLP, 30 Rockefeller Plaza, New York, NY 10112-0228 (US).

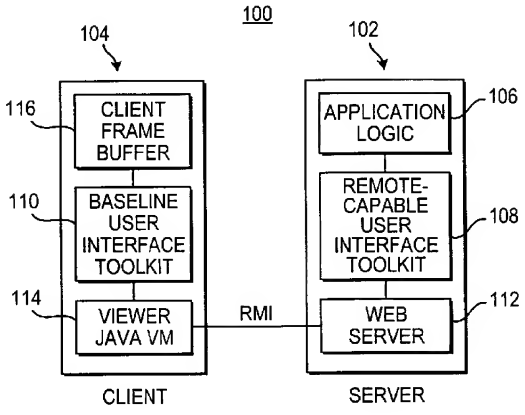
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: DISTRIBUTED COMPUTER SYSTEM USING A GRAPHICAL USER INTERFACE TOOLKIT



(57) Abstract: A distributed computer system having a server and remote client for executing an application on the server. A remote-capable user interface toolkit resides on the server and has remote-capable components that correspond to components of a user interface toolkit which resides on the remote client. The remote-capable components are substantially the same as corresponding components of the user interface toolkit, and interact with the application according to the same application programming interface. However, when invoked by the application, the remote-capable components issue a message to the component on the remote client to perform the corresponding function on the client. A network communication protocol of sending messages between the remote-capable user interface toolkit on the server and the user interface toolkit on the client is thereby generated. The remote-capable components may be created by a code-generating routine which reads in the component of the user interface toolkit, copies the code of the component, and substitutes a portion of the code relevant to performing the function of the component with a portion of code that issues a remote message to a component on a remote client to perform the same function.

WO 01/95107 A2

DISTRIBUTED COMPUTER SYSTEM USING A GRAPHICAL USER INTERFACE TOOLKIT

SPECIFICATION

CROSS-REFERENCE TO RELATED APPLICATION

This application claims priority to U.S. Provisional Patent Application Serial No. 60/210,643, filed on June 9, 2000, entitled "Method and System to Support Rich User Interfaces on Light Clients," and U.S. Provisional Patent Application Serial No. 60/277,498, filed on March 21, 2001, entitled "Thin Client Graphical User Interface Toolkit," both of which are hereby incorporated by reference in their entirety herein.

BACKGROUND OF THE INVENTION

This invention relates to computer systems using distributed user interfaces, and more particularly, to distributed user interfaces using user interface toolkits.

Many approaches have been researched academically and deployed commercially to support a distributed computing paradigm in which the network separates the presentation of the user interface from the application logic. Two approaches are commonly used in both industry and academia: web-based and remote frame-buffer–based. A third approach, distributed user interface toolkits, provides additional advantages, but still has significant drawbacks.

The first approach to distributed computing is one of the most widely deployed approaches to thin-client computing, and uses HyperText Transfer Protocol (HTTP) (See T. Berners-Lee et al., "Hypertext transfer protocol" - HTTP/1.0 RFC1945, 1996) and HyperText Markup Language (HTML) (See T. Berners-Lee et al., "Hypertext markup language" - 2.0 RFC1866, 1995 and D. Conolly et al., "The text/html media type," RFC2854, 2000) for the client with the server, commonly known as the world wide web. The architecture of an application developed using a web-based methodology is depicted in FIG. 1. As illustrated in FIG. 1, a server 10 is in communication with a client 12 over a network. The application logic 14 and the

web server 16 reside on the server 10. A special web application programmer interface 18 (API) is provided to allow the application to communicate with the web server. Typical web API's are CGI (See "The Common Gateway Interface." http://hoohoo.ncsa.uiuc.edu/cgi/overview.html), ISAP (See "ISAPI Extensions Overview." http://msdn.microsoft.com/library/psdk/iisref/isgu9kqf.htm), NSAPI (See NSAPI FAQ. http://developer.netscape.com/support/faqs/champions/nsapi.html), ASP (See "An ASP you can grasp: The ABCs of active server pages." http://msdn.microsoft.com/workship/server/asp/ASPover.asp.), PHP (See "PHP: Hypertext Preprocessor." http://www.php.net), or JSP (See "JavaServer Pages: Dynamically Generated Web Content." http://java.sun.com/products/jsp). HTTP is used to negotiate the transfer of HTML data between the client web browser 20 and the web server 16. The web browser 20 then renders the HTML 21 onto the client frame buffer 22, from which visual presentations are generated on the display of the client. The user may interact with the displayed presentation to send data back to the web server via HTML

One severe limitation of a web-based approach using HTTP/HTML is the "pull-only" data transfer methodology, which prevents the application from generating events. For example, when a user executes a search on a web search engine, the engine must ideally complete the search in its entirety within a few seconds of the request because the user is expecting an immediate response. After the initial page has been displayed, the web search engine cannot notify the user that better results have been found. A second problem is that HTTP is stateless, which makes it difficult for programmers to create even a simplistic notion of persistence between page accesses. In addition, the user interaction is also extremely limited, providing only a handful of the most commonly used interactive functions.

Many attempts have been made to address these problems, including sending entire applications over HTTP (e.g., JAVA™ applets. See JAVA™ Applets. http://java.sun.com/applets), designing browser "plug-ins" that interpret their own language to provide a richer user experience (e.g., Macromedia Flash and Shockwave See Macromedia, Inc. http://www.macromedia.com), creating a 3D world in which the user can navigate (e.g., VRML), and providing an application programmer interface (API) for storing persistent session identification data (e.g., cookies. See D.

Kristol et al., "HTTP state management mechanism," RFC2109, 1997). All these approaches to addressing the problems with HTTP/HTML create new problems.

JAVA™ applets raise numerous security concerns because HTTP is used to transport executable code to the client. Although the byte codes transmitted across the network are in compiled form, JAVA™ decompilers are readily available that will allow any user to have access to the source code of the application. In addition, the use of JAVA™ applets typically violates the thin-client principle of not running any application logic on the client. Flash and VRML define richer languages that have been built with user interactivity in mind, but suffer from the problem that mature browsers for anything other than the Microsoft Windows desktop operating systems are generally not available. HTTP cookies raise numerous security concerns because they permit the server program to write data to the permanent storage device on the client. In addition, HTTP cookies have been the target of severe criticism due to a recent surge in public awareness regarding privacy concerns when using the Internet. These issues make HTTP cookies an unattractive method for programmers to add server-side state to the HTTP protocol.

A second approach to distributed computing involves creating a remote virtual frame buffer on the server, on which the application can draw, and then transporting the resulting raster image to the client. In essence, this approach attempts to bring the server's desktop to the user and thereby permits a full range of user interactivity. Products such as CITRIX™ METAFRAME™ (See http://www.citrix.com/products/metaframe/), INSIGNIA SOLUTIONS™ NTRIGUE™ (See http://www.insignia.com), SCO TARANTELLA™ (See http://www.tarantella.sco.com), GRAPHON™ RapidX (See http://www.graphon.com) and SYMANTEC™ PCANYWHERE™ (See http://www.symantec.com) are among those that have been providing this type of functionality for many years as an extension to the underlying operating system. A recent explosion in the popularity of this approach occurred when AT&T released their cross-platform VNC system to the public free of charge. (See Q. Li et al., "Integrating Synchronous and Asynchronous Collaboration with VNC," *IEEE Internet Computing*, 4(3):26-33, May-Jun 2000.) Microsoft has now made this

capability a standard part of their Windows 2000 operating system (See http://www.microsoft.com/windows2000/technologies/terminal/default.asp).

The architecture of a remote frame buffer based application is illustrated in FIG. 2 for transmission between a server 24 and client 26. The application 28 is typically written using a standard user interface toolkit API 30, such as JAVA™ Foundation Class (hereinafter "JFC") (See "JAVA™ Foundation Classes: Now and the Future" http://java.sun.com/products/jfc/whitepaper.html), Microsoft Foundation Class (See *Microsoft Visual C++ MFC Library Reference.* Microsoft Press, Redmond, WA, 1997), Tk (See J. Ousterhout. *Tcl and the Toolkit.* Addison-Wesley, 1994), or MOTIF (See *Modular Toolkit Environment.* IEEE 1295), and renders onto a remote virtual frame buffer 32. The resulting pixel data 34 is transported across the network using a proprietary protocol, such as ICA (See Citrix Metaframe. http://www.citrix.com/products/metaframe), RFB (See "Microsoft Windows 2000 Terminal Services." http://www.microsoft.com/windows2000/guide/server/features/terminalsvs.asp), or RDP (See T. Ricardson et al., "Virtual network computing," *IEEE Internet Computing*, 2(1):33-38, Jan-Feb 1998) to the client 26. The client viewer 36 receives the pixels and reconstructs the image, and then copies the image onto the client frame buffer 38 for presentation on the client's display.

Although the remote frame buffer approach addresses many of the problems with a web-based approach that uses HTTP/HTML, it also introduces a number of other problems. Whereas the web-based approach using HTTP/HTML is capable of operating reasonably well over relatively low-bandwidth modem network links, the remote frame buffer approach demands high-bandwidth connections. This is because the remote frame buffer approach is essentially sending a video stream of computer-generated graphics from the server to the client.

Although the use of advanced lossy video compression algorithms (e.g. MPEG (See ISO/IEC JTC1/SC2/WG11.MPEG. *ISO*, Sept. 1990)) has been proposed (T. Ricardson et al., "Virtual network computing," *IEEE Internet Computing*, 2(1):33-38, Jan-Feb. 1998), implementation of such techniques may presents several technical difficulties. For example, real-time compression of MPEG streams usually requires special hardware that can only handle one or two streams at a time, thereby

eliminating the possibility of using the remote frame buffer approach on a current shared server. In addition, the use of lossy compression techniques introduces unwanted compression artifacts into the display, reducing the system's usability, particularly when working with text and detailed graphics.

The existence of server-side state and asynchronous event generation by the server permits the remote frame buffer approach to provide a rich level of user interactivity that a web-based approach using HTTP/HTML cannot. However, there is a practical limitation caused by network latency. For example, such problems may arise in connection with the display of a mouse pointer on a typical client "viewer," i.e., the remote frame buffer analogue of the web browser. Under certain circumstances, the client viewer may display two mouse pointers. One mouse pointer represents where the cursor should be pointing, and is tied to the local mouse. A second mouse pointer, which typically lags behind the first mouse pointer, displays where the mouse position is on the server. When a remote frame buffer system is run on anything other than a high-speed LAN connection (e.g., 100 megabit per second over category 5 cabling), there is always a noticeable difference in position between the client (virtual) and server (real) mouse positions. On a slow modem link (e.g., 56.6 kilobit per second transmission over a standard telephone line), this makes highly interactive user interfaces difficult to control, and, in extreme cases, may even make the system unusable.

A third approach to distributed computing is distributed user interface toolkits, which address the issues that arise when employing web-based HTTP/HTML and remote frame buffer approaches by allowing a server to manipulate user interface toolkit components directly on the client. The server can create, modify, and delete any of the components available in the distributed toolkit as if it were working with a local application. This approach is analogous to an implementation of a remote frame buffer with an extremely efficient, lossless compression algorithm. Instead of sending pixel data rendered on the server across the network, the distributed user interface toolkit sends the semantics necessary to render that pixel data on the client. In addition, since the mouse is handled locally on the client, there is no additional perceived latency beyond that caused by the processing that is necessary to service users requests when the application is running locally.

The current approaches to distributed user interface toolkits have several disadvantages. The X Window System (See R. Scheifler et al., "The X Window System," *ACM Trans. on Graphics*, 5(2):79-109, April 1986), for example, transports low-level drawing commands. If a high-level user interface toolkit is used with X, the high-level user interface toolkit commands (e.g., draw button) are actually translated into low-level commands (e.g., lines and rectangles) before being transmitted across the network. Another disadvantage is that the X Window System stores state on the client computer that is presenting the output to the end user. Consequently, it is very difficult to "share" X Window System sessions between multiple users, and if the X Window System running on the client computer fails, the user session is lost. It is for these reasons that a remote virtual frame buffer system, such as VNC, is often employed to transport an X Window System desktop from a UNIX server to an X Window System viewer running on a UNIX workstation, rather than relying on the built-in networking facilities of X.

There is therefore a need in the art for a distributed user interface that runs the application logic on the server computer but which also allow the server computer to asynchronously generate events and transmit them to the server. There is also a need for a distributed user interface that allows relatively sophisticated graphics without requiring high-bandwidth connections. In addition, there is also a need for a distributed user interface which is easily implemented and does not require the creation of a new protocol of communication.

SUMMARY OF THE INVENTION

It is an object of the invention to provide a distributed computer system which is compatible with toolkits of well-known programming languages and implicitly creates a protocol of network communication.

It is another object of the invention to provide a distributed computer system that does not require high-bandwidth to operate and which allows a high degree of user interactivity.

These and other objects of the invention which will become apparent with respect to the disclosure herein, are accomplished by a novel distributed computer system having at least one server and one remote client to execute an

application entirely on the server, wherein the application so configured to interact with a user interface toolkit according to an application programming interface. A user interface toolkit is provided, which resides on the remote client and has at least one component configured to perform a function on the remote client. In an exemplary embodiment, JAVA™ Foundation Class is the user interface toolkit which has a plurality of components known as the Swing component class.

A remote-capable user interface toolkit resides on the server. The remote-capable user interface toolkit has at least one remote-capable component which interfaces with the application according to the same application programming interface as the user interface toolkit and which is configured to generate a message to perform the respective function of the corresponding component in the user interface toolkit in response to an invocation by the application. The remote-capable component is otherwise identical to the component.

The protocol of communication between the remote-capable component of the remote-capable user interface toolkit on the server and the component of the user interface toolkit on the client comprises the transmitting of messages by the remote-capable component invoked by the application.

The component in the user interface toolkit may be configured to render a graphical item and the remote-capable component may be configured to generate a command to render a graphical item. Similarly, the server may be configured to communicate the message to the user interface toolkit on the remote client to render a graphical item in response to the invocation by the application. The component of the user interface toolkit on the remote client may be configured to render the graphical item in response to the message.

The component in the user interface toolkit may be configured to install an event handler and the remote-capable component may be configured to generate a command to install an event handler. Similarly, the server may be configured to communicate the message to the user interface toolkit on the remote client to install an event handler, and the component of the user interface toolkit on the remote client may be configured to install the event handler in response to the message.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a simplified block diagram of a prior art system.

FIG. 2 is a simplified block diagram of a second prior art system.

FIG. 3 is a simplified block diagram of the system in accordance with the invention.

FIGS. 4(a) – 4(c) illustrate prior art user interface toolkit components.

FIG. 5 illustrates a user interface in accordance with the invention.

FIGS. 6(a) – 6(b) illustrate an application as rendered on a client buffer in accordance with the invention.

FIG. 7 illustrates another application as rendered on a client buffer in accordance with the invention.

FIG. 8 illustrates a further application as rendered on a client buffer in accordance with the invention.

FIG. 9(a) illustrates executable code in accordance with the invention.

FIG. 9(b) illustrates prior art executable code.

DETAILED DESCRIPTION OF THE EXEMPLARY EMBODIMENTS

The architecture of a distributed user interface system 100 in accordance with the invention is illustrated in FIG. 3 and includes a server 102 and a client 104. The application logic 106 resides on the server 102. A novel remote-capable user interface toolkit 108 resides on the server 102 and a baseline user interface toolkit 110 resides on the client 104. As will be described below, the remote-capable user interface toolkit 108 has components which correspond to components in the baseline interface toolkit 110, but which issue remote messages rather than execute graphical functions. These messages are interpreted by a server JAVA™ virtual machine 112 ("server VM") that transmits the commands across the network to the client 104. A client viewer JAVA™ virtual machine 114 ("client viewer") translates the messages issued by the remote-capable user interface toolkit 108 into function calls of the baseline interface toolkit 110, which are rendered on the client frame buffer 116. It is noted that according to another exemplary embodiment, using a programming language other than JAVA™, the system is implemented without a virtual machine.

The distributed user interface system 100 makes use of visual components (often called widgets or controls) that are gathered together in libraries that are usually referred to as user interface toolkits. The exemplary embodiment utilizes JFC as the baseline graphical user interface toolkit 110. (The JAVA™ language specification, B Joy et al., *The JAVA Language Specification*, Addison Wesley, 2d Ed., 2000 and http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html and JAVA™ virtual machine specification, T. Linde et al., *The Java Virtual Machine Specification,* Addison Wesley, 2d Ed., 1999 and http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html, and "JAVA™ Foundation Classes: Now and the Future" http://java.sun.com/products/jfc/whitepaper.html, have been incorporated by reference in their entirety herein.) JFC has been utilized in the exemplary embodiment because of its ability to create cross-platform compatible graphical user interfaces. However, it is noted that the system and methods described herein are also compatible with any available toolkit.

A user interface toolkit, as understood in the specification and claims, is computer code which provides an application programming interface that (1) renders at least one graphical component related to user interaction in response to an invocation by the application, and (2) generates an event coupled to the graphical component in response to user interaction with that graphical component. These functions are described in greater detail herein. First, a toolkit has the ability to draw a frequently-used, graphical components on a user display as commanded by an application running on the computer. Each graphical component is concerned with an aspect of user interaction, and therefore visually provides the user with one or more selectable options as well as a manner of making a selection. Typical components in a toolkit draw graphical items such as buttons, scrollbars, menus, text fields, and the like. In rendering the graphical component, the toolkit may include commands to display a plurality of shapes, colors, and text. The toolkit is configured to interact with the application according to an application programming interface. For example, the toolkit receives an invocation, or call, from the application to draw graphical components at certain times during the operation of the application. In the exemplary embodiment, JFC has a well-defined application programming interface.

It is noted that a toolkit may comprise a single component, such as a button, or it may generate a plurality of multiple components. JFC, for example, provides many components bundled together in a component set referred to as "Swing." (See "The Swing Component Galley" http://java.sun.com/products/jfc/tsc/articles/component_gallery/index.html, which is incorporated by reference in its entirety herein.) Exemplary components of Swing include "JButton," illustrated in FIG. 4(a), "JCheckbox," illustrated in FIG. 4(b), and "JRadioButton," illustrated in FIG. 4(c). JButton is a commonly used component that may be selected, i.e., "clicked," by the user. JCheckbox is an image including a group of items and provides the user with the ability to select or de-select one or more of these items. Similarly, JRadioButton is an image including a group of buttons. In contrast with JCheckbox, JRadioButton allows only one button at a time to be selected. (According to convention, selecting a new button in JRadioButton will simultaneously select the new button and de-select a previously selected button.)

A second, related feature of a toolkit is the ability to generate an event based on a user response, if any, to the component rendered on the user display. The toolkit is thus able to provide a link between (1) the syntax of the user interaction (e.g., typing a character or pressing a mouse button), and (2) the semantics necessary to carry out the function commanded by that user interaction (e.g., closing a text window.) The toolkit includes an event handler that "listens" (i.e., waits), for a specific user interaction to occur, and then generates an event when that interaction occurs. (Each event may be represented by an object that gives information about the event and identifies the event source.). For example, a button (e.g., JButton), may be configured to wait for the user to click the button (i.e., press a mouse key while positioned over the button). When the user clicks the button, the toolkit generates an event. In this case, the result may be that a toolkit text window is automatically closed when the event listener detects an event triggered by the button component.

It is further noted that the procedures described herein are applicable to a user interface toolkit which "renders" an item to the user which may be graphical, audio, tactile, olfactory or other sensory modality, that may be coupled with the generation of an event in the nature of a user interface.

The toolkit, as described above, interacts with the application according to an application programming interface. In addition to receiving commands to draw graphical items, the toolkit generates events, which are usually associated with components. These events are then conveyed to the application according to the application programming interface, which enables the application to take some action based on the events generated by the user. JFC, as implemented in the exemplary embodiment, interacts with the application according to a well-defined application programming interface from the standpoint of conveying events to the application.

The user interface toolkit provides an abstraction layer for drawing the graphical items and generating events, by using the low-level drawing and interaction routines made available to programmers by the graphics subsystem that is usually bundled with the operating system. This abstraction allows programmers to quickly create commonly used visual components, such as buttons, scrollbars, menus, and text fields. End users also benefit, since most of the applications they run on a particular operating system will have roughly the same "look and feel" because the applications are all built out of components from the same user interface toolkit.

The typical implementation of a user interface toolkit, such as JFC, is on a system in which the application logic execution and the user interface presentation occur on a single computer. The tight binding of the user interface toolkit to the underlying graphics subsystem allows this type of implementation. However, the use of the toolkit when creating distributed applications in which the application logic execution and user interface presentation occur on different computers may present significant challenges.

With continued reference to FIG. 3, the distributed user interface system 100 is configured to work with any toolkit, as described above, which interfaces with application logic 106 and has the capability to draw graphical components and generate or respond to events. The system 100 includes a remote-capable interface toolkit 108, which resides on the server 102. As described above, JFC was used as the baseline user interface toolkit 100 implemented on the client 104, and "Remote JAVA™ Foundation Classes" (RJFC) was created as a remote-capable version of JFC. JFC was selected as a baseline interface toolkit 108 for the exemplary

embodiment because of its familiarity to programmers and richness in functionality. JFC API is extremely complex, and includes over 600 individual source files, each providing between 10 to 100 methods for the programmer to use.

The remote-capable version of the toolkit 108 is a toolkit which appears to the application logic 106 as a local toolkit for drawing graphical components and generating events. However, when invoked by the application logic 106, the remote-capable user interface toolkit issues a remote process invocation, such as JAVA™ RMI, for drawing the graphics or generating events on the remote client 104. More particularly, RJFC has one or more components that are substantially identical to components in the corresponding baseline toolkit 110, JFC. Thus, there is a one-to-one correspondence between JFC components and RJFC components. A significant difference between these components, however, is that a JFC component, when invoked, locally performs a particular function (e.g., it draws a button on the local VM or it generates an event, as described above). In contrast, the corresponding RJFC component is configured to send a message to perform that same function, i.e., drawing a graphical item or generate an event, which is transmitted to the remote client 104. Alternatively, if the RJFC component is an event handler, it is configured to receive a remote signal concerning the occurrence of an event. Thus, the application programming interface of the RJFC 108 tracks the design pattern and functionality of the application programming interface of the standard JFC 110 as closely as possible, with the exception that the presentation displays on a remote client 104 or the event is generated at the remote client 104, rather than on a local frame buffer.

RJFC components are generated automatically from the JFC source code by a "code generator" application. Since the source code to JFC is readily available, a code generator reads the JFC source and produces a RJFC component for each JFC component. In the exemplary embodiment, a modified version of JAVA™ Doclet has been used to read the JFC source code in the JAVA™ programming language and to produce RJFC code (also in the JAVA™ programming language) for each respective JFC component. The Doclet is a publicly available tool that was designed to read in source code and automatically generate documentation. In

accordance with the invention, the Doclet has been modified to generate source code in the JAVA™ programming language rather than documentation.

The procedure of using a code generator provides a great degree of automation and flexibility because the components of the remote-capable toolkit 108 do not have to be separately and individually programmed. In addition, the remote-capable toolkit components do not have to be rewritten if the underlying toolkit is modified. This approach may be used to generate different versions of the remote-capable toolkit system for various implementations and releases of the JAVA™ SDK, making it possible to handle a broad range of supported JAVA™ VM's.

Another advantage of creating the remote-capable toolkit by use of a code generator is that the application programming interface of the remote-capable user interface toolkit 108 is *implicitly* identical to the application programming interface of the baseline interface toolkit 110 which resides on the client 104. Consequently, manipulation of the RJFC components (e.g., changing the text of a label) and association of event handlers by the application logic 106 is syntactically identical to the JFC API. Although each RJFC component has an actual associated JFC component that resides in the client viewer's memory space, the application logic 106 which resides on the server 102 interacts with the remote client 104 by making calls on the RJFC components on the server 102 alone. Since the actual JFC components that are used to create the display on the client frame buffer are hidden from the application logic 106, the application logic 106 is not modified to operate in the distributed environment. Since RJFC components track the JFC API and follow the Sun JAVA™ Beans standard (See G. Voss "JAVA™ Beans" http://developer.java.sun.com/developer/onlineTraining/Beans/Beans1/simple-definition.html), they may also be easily used in graphical user interface builders such as SUN FORTE™ for JAVA™ (See http://www.sun.com/forte/ffj), BORLAND™ JBUILDER™ (See http://www.borland.com/jbuilder) and WEBGAIN™ VISUALCAFE™ (See http://www.webgain.com/products/visual_cafe).

The procedure for distributed processing through a server and a remote client proceeds as follows. The application logic 106 is executed entirely in the server 102. The application logic 106 is configured by the programmer to interact with the user interface toolkit according to an application programming interface. A user

interface toolkit, as defined above, comprises one or more components that perform several functions: the component may render a graphical item when invoked by an application, and may generate an event in response to a user interaction with that graphical item. In the exemplary embodiment, the baseline user interface toolkit 110 may be JFC, and the components may be the Swing component set.

An early stage in the procedure may be to provide the user interface toolkit 110 on the remote client 104 such that the component is configured to perform the function on the remote client 104. In the exemplary embodiment, the JFC components are provided on the remote client and are able to render the graphical items on the client frame buffer 116 and generate events at the remote client VM 114.

A subsequent stage may be to provide a remote-capable user interface toolkit 108 on the server 102. The remote-capable user interface toolkit 108 is provided by creating at least one remote-capable component which is configured to interact with the application logic 106 according to the same application programming interface as the baseline user interface toolkit 110 and which is configured to generate a remote message to the component on the remote client 104 to perform the respective function on the remote client 104. According the exemplary embodiment, the remote-capable user interface toolkit 108 is referred to as RJFC wherein each component of RJFC is syntactically identical to each component in JFC, with the except that the portion of the code in the remote-capable component has been substituted with a portion of code that generates a remote message to the JFC component to perform the same function.

A next stage in the procedure may be to invoke the remote-capable user interface toolkit 108 by the application logic 106 according to the application programming interface to perform a function. At a subsequent stage, the remote-capable user interface toolkit 108 generates a remote message to perform the function invoked by the application logic 106. Since there is a one-to-one correspondence between the JFC component and the RJFC component, a protocol of communication between the RJFC component and the JFC component is implicitly defined. This protocol of communication comprises the transferring of massages to perform JFC functions, and such messages are issued in the manner in which the JFC toolkit would

normally perform functions on a single computer. Therefore, there is no need to specifically create a protocol of communication.

The message may be communicated between the remote-capable user interface toolkit on the server and the user interface toolkit on the remote client in a subsequent step. In the exemplary embodiment, this communication between the server 102 and the client 104 uses remote method invocation (RMI) (See S. McPherson, "JAVA™ Servlets and Serialization with RMI," http://developer.java.sun.com/developer/technicalArticles/RMI/rmi/.

A later stage may be to perform the function on the remote client by the component of the user interface toolkit in response to the message. Thus, when an RJFC component is instantiated, modified, or deleted on the server 102 by the application logic 106, the RJFC toolkit 108 transparently informs the client viewer 114 of the event that has occurred. The client viewer 114 reacts to the message by performing the exact same action on the client viewer 114 that would have occurred on the server 102 if the JFC API were used.

For example, the standard JFC component JButton serves as the basis for the RJFC component RJButton. (Whereas JButton renders a button, RJButton sends a message to remotely render a button.) If the server 102 requests that a new RJButton be created, the RJFC toolkit 108 would generate a message which the server VM 112 transmits to the client viewer 114. The client viewer 114 receives the message and then creates a JButton using the standard JFC API 110, thus causing the actual button to be rendered on the client frame buffer 116. Similarly, when the server 102 installs an event handler into a RJFC component, the server 102 communicates with the client viewer 114, using RMI, to install a proxy JFC event handler into the associated JFC component that is being displayed on the client frame buffer 116.

One key performance optimization in the RJFC Protocol is the use of a component-generating object, referred to in the exemplary embodiment as "RJFCFactory," that resides in the client viewer's memory space. RJFCFactory is a piece of code that defines what components the application logic 106 can cause to appear on the client 104. This code for RJFCFactory is automatically generated by the code generator, described above. The code generator reads the JFC source code

and creating a remote-capable method for each baseline JFC method. RJFCFactory performs two actions: (1) it creates JFC components in the client viewer's memory space and (2) transmits to the server 102 a reference to RJFCFactory. (In the exemplary embodiment implemented in the JAVA™ programming language, RJFCFactory extends UnicastRemoteObject and implements an interface that extends Remote.) When a client viewer 114 connects to a server 102, the client viewer 114 passes the reference to the RJFCFactory during a display registration method implemented on the server 102. Once the server 102 has received the reference to RJFCFactory, the server 102 can do the following: (1) transmit commands to RJFCFactory to create JFC components that reside in the client viewer's memory space and (2) receive a remote reference to the associated RJFC wrapper object from the client 104. This procedure eliminates the need to create a serialized object in the server's memory space, subsequently send the serialized object to the client 104, and then send a remote reference to the wrapper object back to the server 102. Test measurements show that a RMI call as described above consumes approximately five Ethernet packets whereas sending a serialized JButton consumes more than ten times that number.

The protocol for the remote-capable user interface toolkit 108 in accordance with the invention accomplishes event handling using a similar methodology. The following protocol may be followed to allow the client 104 to transmit client-generated events to the server 102: If an event handler is installed into a RJFC component on the server 102, the server 102 may transmit a simple message to the client viewer 114, using RMI, that tells the client viewer 114 to install a proxy event handler in the associated JFC component. The proxy event handler on the client 104 makes a call to the server 102 whenever a new event is generated on the client side. The actual semantics of the event handler, as defined by the application logic 106, is executed on the server 102 when the server 102 receives the RMI call from the client 104. Similarly, the following protocol may be followed for transmitting server-generated events to the client 104: The server 102 retains the reference to the RJFC component returned by the RJFCFactory after the display initialization is completed. When the server 102 generates events, it transmits a command to the client 104 with a remote reference to the RJFC component. This protocol enables the server 102 to

asynchronously generate events at will, i.e., without requests from the user at the remote client.

An exemplary RJFC viewer 200, as illustrated in FIG. 5, provides a context in which the application logic 106 which resides on the server 102 can manipulate the client frame buffer 116. The viewer is an application, which may be hand-coded, that uses the baseline interface toolkit 110, e.g., JFC, and emulates the functionality found in a typical thin-client system. The user of the system invokes the client viewer 114, at which point a JFrame window 202 is created with a form 204 that allows the user to connect to a server 102. Once a connection is established, a second JFrame window 206 is created for the server 102 to manipulate remotely. The server 102 may also request that additional windows be created by asking for dialogue boxes using the RJFC API. FIGS. 6-8 illustrate several small applications being run in the client viewer 114. FIGS. 6(a)-6(b) illustrate a "notepad" application 210 being run on the client viewer 114 as rendered by the JFC toolkit 110 in response to commands from the RJFC toolkit 108 residing on the server 102 as described herein. The notepad application 210 implements several of the JFC Swing components, such as JButton 212, JScrollPane 214, JPopUpMenu 216, and JOptionPane 218.

Similarly, FIG. 7 illustrates a simple web browser application 230 which conducts searches for web pages in response to user requests, as is well known in the art. As described above, the invention provides the capability to transmit server-generated events to the client 104: In the exemplary embodiment, RJFCFactory object resides on the client 104, and it sends an RJFCFactory reference to the server 102 during the display initialization. The server 102 retains this reference to the RJFC component. When the server 102 generates events, it transmits a command to the client 104 with a remote reference to the RJFC component. This protocol enables the server 102 to asynchronously generate events at will, i.e., without requests from the user at the remote client. As illustrated in FIG. 7, the application may be a web browser. The protocol according to the invention provides a substantial benefit over the HTTP/HTML web browser applications. For example, when the user requests a web search, the server sends commands to the client to display initial results of the search. (This is similar to the HTTP/HTML system.) However, in accordance with the invention, the server 102 may continue to search for additional

results. When these newer results are found, the server 102 generates an event, and is able to transmit an RJFC command to the client with a RJFCFactory reference to the appropriate JFC component on the client 104 to display the results on the client frame buffer 114. This process may proceed asynchronously to update the search results without any further inputs from the user.

FIG. 8 illustrates a simple spreadsheet application 240, each of which is rendered on the client buffer 116 in response to commands generated by the RJFC toolkit 118 in accordance with the invention.

EXAMPLE

An example of code for creating a simple "notepad" application written using the RJFC API is shown in FIG. 9(a), and a baseline, i.e., non-network-aware version of the code in JFC API is shown in FIG. 9(b). The code generator was configured such that the resulting RJFC API has a one-to-one correspondence to JFC components. In the exemplary embodiment, a capital "R" (indicative of the remote-capable functionality) is prepended to the name of the toolkit component being referenced. The significant difference between the non-network-aware JFC application of FIG. 9(b) and the remote-capable RJFC application of FIG. 9(a) is that the JFC code calls "new" to instantiate a component, whereas the RJFC code makes a remote method invocation to an RJFCFactory object (as described above) which resides in the client viewer's memory space.

The distributed user interface in accordance with the invention was compared with the web-based thin-client approach using HTTP/HTML as illustrated in FIG. 1 and the remote frame buffer approach as illustrated in FIG. 2, above. The implementation of HTTP/HTML consumes very little bandwidth because HTML represents a presentation's semantics at an extremely high level. While this causes a relatively small amount of information to be transported, this approach suffers from the problem that HTTP was not designed for implementing remote applications, but rather for sharing static data.

In contrast, the remote frame buffer approach operates on the premise that compatibility with existing applications is paramount at the expense of network bandwidth. This is because many of the remote frame buffer implementations were

designed for corporate or lab network environments whose administrators are trying to move users away from desktop computers to a thin-client subsystem with a lower total cost of ownership.

The RJFC distributed user interface toolkit in accordance with the invention combines the benefits of both approaches without their performance and usability issues by transmitting the high-level semantics of a display using a standard toolkit API. The network bandwidth consumed by RJFC is closer to that of the web-based approach using HTTP/HTML than that of the remote frame buffer approach, while permitting rich user interaction without artificially introduced latency. TABLE 1 is a comparison of the bandwidth consumed by RemoteJFC and the AT&T VNC remote frame buffer system. TABLE 1 shows the number of Ethernet packets transmitted over the network by VNC and by the distributed user interface system 100 using the remote-capable user interface toolkit 108 when simple operations were performed in a notepad application similar to that illustrated in FIG. 9(a). In the VNC system, a large number of packets transmitted were due to movement of the mouse by the user. The amount of mouse movement by a novice user may be significantly greater than the movement of a more experienced user. Since the amount of user experience may affect the comparison, both VNC novice and VNC expert data is included in TABLE 1. (Since a web-based method using HTTP/HTML would not be able to provide the same level of user interactivity, this approach was omitted from TABLE 1. It is noted, however, that the Client Connection cost of a web-based "notepad" application is approximately 10 packets.)

Operation	RJFC	VNC Expert	VNC Novice
Client Connection	620	450	450
Load File	85	860	2600
Popup About Dialog	24	70	1500
Close About Dialog	32	85	630
Maximize Window	8	690	1000
Scroll to Bottom of Page	0	420	1700

TABLE 1

Thin-client systems need some kind of software browser or viewer that must reside in permanent storage on the client computer. Because the web-based approach and the novel remote-capable user interface toolkit approach both transmit high-level information across the network, the size of the client software package is therefore larger than that of the VNC viewer.

The size of a typical web browser download is about 25 megabytes, as compared to the VNC viewer which can be about 110 kilobytes. The client viewer 114 lies somewhere in between: the RJFC library adds 2.5 megabytes to the underlying JAVA™ runtime environment, which can vary in size from 30 to 15 megabytes. In addition, the VNC viewer memory image when attached to an 800x600 desktop consumes 1.5 megabytes of RAM, whereas both the web browser and client viewer 114 require approximately ten times that amount. This also results in faster startup times for the VNC viewer than a web browser or the client viewer 114.

Overall, the remote frame buffer approach is much "thinner" than the web-based and RemoteJFC approaches and is capable of running on less powerful hardware, but requires much more network bandwidth to operate effectively.

An appendix which sets forth the computer code for the significant code routines is appended hereto and incorporated by reference in its entirety herein. In particular, `Doclet` is the main routine for the code generator program, as described above. `Doclet` is only executed once to create the RJFC library. The `Viewer` routine resides on the client. It is the vehicle through which the RJFC application displays its output. `NotepadServer` is a demo RJFC application. It resides on the server, and has the control logic that can generate the graphics shown in FIGS. 5-9, above. `RJFCServer` is a routine that executes on the server, and allows a client viewer to register a display with the server and hence allow the server to control the display. `RJFCFactory` is a routine that defines what elements the application can cause to appear on the client. This code is automatically generated by the code generator by reading the Swing source code and creating a RJFC method for each Swing method. This code executes on the client and passes references back to the server.

It will be understood that the foregoing is only illustrative of the principles of the invention, and that various modifications can be made by those skilled in the art without departing from the scope and spirit of the invention.

```
/**
 * Copyright (c) 2001 Computer Graphics & User Interfaces Lab
 * Columbia University
 */

import com.sun.javadoc.*;
import java.util.*;
import java.io.*;


public class Doclet {

    public static final String COPYRIGHT =
        "/**\n"+
        "* Copyright (c) 2001 Computer Graphics & User Interfaces Lab\n"+
        "* Columbia University\n"+
        "*/\n\n";
    private static RootDoc rd;
    private static Writer impl; /* write to implementation class */
    private static Writer intr; /* write to interface */
    public final static String PACKAGE = "edu.columbia.cs.cgui.rjfc";
    public final static String PATH = "./edu/columbia/cs/cgui/rjfc/";
    public static boolean test; //TESTING PURPOSES
    public static final String VARSUFFIX = "var";

    //to keep track of which methods we have already printed
    //this is to avoid reprinting methods that were overriden
    public static HashSet hash = new HashSet();


    public static boolean start(RootDoc root) throws IOException{

        rd = root;
        test = true; //TESTING PURPOSES

        /* INSERT FILENAME HERE
           Should read public interface blah blah
        */

        //populate hash with all methods from Object so that we
        //don't try to override them. because javadoc is a stimpy
        //we have to do this manually
        /*
          //for some reason root.classNamed(..) returns null sometimes
          ClassDoc object_class = root.classNamed("java.lang.Object");
          System.err.println("obj class: "+object_class);
          MethodDoc[] object_methods = object_class.methods();
          for(int i=0;i<object_methods.length;i++) {
          hash.put(object_methods[i].name()+object_methods[i].signature(),"");
          }
        */


        /*
         * We added these to the hash so that we don't overwrite one of
         * Object's original methods with another method that throws
         * RemoteException
         */
        hash.add("clone()");
        hash.add("equals(java.lang.Object)");
        hash.add("finalize()");
        hash.add("getClass()");
        hash.add("hashCode()");
        hash.add("notify()");
        hash.add("notifyAll()");
```

```
hash.add("toString( )");
hash.add("wait()");
hash.add("wait(long)");
hash.add("wait(long, int)");


ClassDoc[] classes = root.classes();
if (classes.length <= 0)
    return true;
//goes through classes
//ONLY LOOKING AT FIRST CLASS, IGNORING INNER CLASS
for (int i = 0; i < 1; ++i) {
    if (classes[i].isInterface())
        return true;
    if (!classes[i].isPublic())
        return true;
    if (!(classes[i].superclass().containingClass() == null))
        return true;

  String className = "";
  String origName = "";
  StringTokenizer pathfinder = new StringTokenizer(classes[i].toString() , ".");
  String path ="/";
  String extrapackage = "";
  pathfinder.nextToken(); //bounce the first token
  while (pathfinder.countTokens()>1) {
      String newToken = pathfinder.nextToken();
      path += newToken + "/";
      extrapackage += "." + newToken;
  }
  System.out.println(path);

  StringTokenizer st = new StringTokenizer(classes[i].toString(), ".");
  while (st.hasMoreTokens()) {
    className = st.nextToken();
  }
  origName = className;
  className = "R" + className;
  //ERROR CHECKING
  System.out.println("***PROCESSING " + origName);

  //Creates file in a subdirectory
  intr = new FileWriter(PATH + path + className + ".java");
  impl = new FileWriter(PATH + path + className + "Impl.java");

  intr.write(COPYRIGHT);
  impl.write(COPYRIGHT);

  /* Name the package */
  intr.write("\npackage "+ PACKAGE + extrapackage +";\n\n"); //need to finish
  impl.write("\npackage "+ PACKAGE + extrapackage +";\n\n"); //need to finish

  /* import packages */
  intr.write("import java.rmi.RemoteException; \n");
  impl.write("import java.rmi.RemoteException; \n");
  intr.write("import java.rmi.Remote; \n\n\n");
  //impl.write("import " + className + ";\n");
  impl.write("import java.rmi.server.UnicastRemoteObject;\n\n\n");
  intr.write("import edu.columbia.cs.cgui.rjfc.event.*;");
  impl.write("import edu.columbia.cs.cgui.rjfc.event.*;");
  //impl.write("import javax.swing.*; \n");
  //intr.write("import javax.swing.*; \n");
  if(classes[i].isPublic()) {
      impl.write("import " + classes[i].toString() + ";\n\n");
      intr.write("import " + classes[i].toString() + ";\n\n");
```

```
            }
            /* start interface here */
            intr.write("public interface "+className+" extends "+
                        (getExtendableName(classes[i]).equals("UnicastRemoteObject")?"Remote":g
etExtendableName(classes[i]))+"{\n");
            impl.write("public " +
                        //(classes[i].isAbstract()?"abstract ":"") +
                        "class "+className+"Impl extends " +
                        getExtendableName(classes[i])+
                        (getExtendableName(classes[i]).equals("UnicastRemoteObject")?"":"Impl")
 +  " implements "
                        + className + "{\n");

            //String origVar = "j" + origName.substring(1);
            String origVar = "data";

            intr.write("\n\n\t// Constructors:\n");
            impl.write("\n\n\t// Constructors:\n");

            printConstructors(classes[i], origVar, origName);

            intr.write("\n\n\t// Fields:\n");
            impl.write("\n\n\t// Fields:\n");
            //impl.write("\tprotected " + origName + " " + origVar + ";\n");

            if (getExtendableName(classes[i]).equals( "UnicastRemoteObject") ) {
                impl.write("\tprotected " + origName + " " + origVar + ";\n");
                impl.write("\tprotected java.util.Hashtable hash;\n");

            }

            origVar = "(("+origName+")"+origVar+")";

            printFields(classes[i], origName);

            intr.write("\n\n\t// Methods:\n");
            impl.write("\n\n\t// Methods:\n");

            if (getExtendableName(classes[i]).equals("UnicastRemoteObject")) {
                intr.write("\n\tpublic "+ origName +" getRJFCWrappedData() throws RemoteExcept
ion;\n");
                impl.write("\n\tpublic "+ origName +" getRJFCWrappedData() throws RemoteExcept
ion { \n\t\t\treturn " + origVar + ";\n\t\t}\n");

                intr.write("\n\tpublic void setData(" + origName + " data) throws RemoteExcept
ion;\n");
                impl.write("\n\tpublic void setData(" + origName + " data) throws RemoteExcept
ion{ \n\t\t\tthis.data = data;\n\t\t}\n");
                //intr.write("\n\tprotected void finalize();\n");
                impl.write("\n\tprotected void finalize() throws Throwable{\n\t\t"+
                           "super.finalize();\n\t\t"+
                           "hash.remove(this);\n\t}\n");
                intr.write("\n\tpublic void setHash(java.util.Hashtable hash) throws RemoteExc
eption;\n");
                impl.write("\n\tpublic void setHash(java.util.Hashtable hash) throws RemoteExc
eption {\n\t\t" +
                           "this.hash = hash;\n\t}\n");

            }

            //if (origName.equals("JPanel")) {
                /* create addRComponent(RJComponent)
                 * addRComponent(RJComponent, int)
                 * addRComponent(RJComponent, Object)
                 * addRComponent(RJComponent, Object, int)
```

```
                * addRCompo    t(String, RJComponent)
                * serverhide();
                * removeRComponent();
                */
          /*   System.out.println("Processing RJPanel: special case\n--");

               intr.write("\n\n\t// Special Methods for RJPanel:\n");
               impl.write("\n\n\t// Special Methods for RJPanel::\n");

               intr.write("\tpublic void addRComponent(RJComponent toAdd) throws RemoteExcept
ion;\n");
               impl.write("\tpublic void addRComponent(RJComponent toAdd) throws RemoteExcept
ion{\n\t\t" +
                          "data.add((javax.swing.JComponent) hash.get(toAdd));\n\t}\n");
               intr.write("\tpublic void addRComponent(RJComponent toAdd, int i) throws Remot
eException;\n");
               impl.write("\tpublic void addRComponent(RJComponent toAdd, int i) throws Remot
eException{\n\t\t" +
                          "data.add((javax.swing.JComponent) hash.get(toAdd), i);\n\t}\n");
               intr.write("\tpublic void addRComponent(RJComponent toAdd, Object constraints)
 throws RemoteException;\n");
               impl.write("\tpublic void addRComponent(RJComponent toAdd, Object constraints)
 throws RemoteException{\n\t\t" +
                          "data.add((javax.swing.JComponent) hash.get(toAdd), constraints);\n
\t}\n");
               intr.write("\tpublic void addRComponent(RJComponent toAdd, Object constraints,
 int i) throws RemoteException;\n");
               impl.write("\tpublic void addRComponent(RJComponent toAdd, Object constraints,
 int i) throws RemoteException{\n\t\t" +
                          "data.add((javax.swing.JComponent) hash.get(toAdd), constraints, i)
;\n\t}\n");
               intr.write("\tpublic void addRComponent(String name, RJComponent toAdd) throws
 RemoteException;\n");
               impl.write("\tpublic void addRComponent(String name, RJComponent toAdd) throws
 RemoteException{\n\t\t" +
                          "data.add(name, (javax.swing.JComponent) hash.get(toAdd));\n\t}\n")
;
               intr.write("\tpublic void removeRComponent(RJComponent data) throws RemoteExce
ption;\n");
               impl.write("\tpublic void removeRComponent(RJComponent data) throws RemoteExce
ption{\n\t\t" +
                          "javax.swing.JComponent toRemove = (javax.swing.JComponent) data.ge
tRJFCWrappedData();\n\t\t" +
                          "remove(toRemove);\n\t}\n");
               intr.write("\tpublic void serverHide() throws RemoteException;\n");
               impl.write("\tpublic void serverHide() throws RemoteException{\n\t\t" +
                          "javax.swing.JFrame jf = (javax.swing.JFrame) javax.swing.SwingUtil
ities.getAncestorOfClass(javax.swing.JFrame.class, data);\n\t\t"+
                          "System.out.println(jf);\n\t\t"+
                          "jf.hide();\n\t}\n");
          } //JPanel Special Case
          */
          printMethods(classes[i], origVar, origName);

          intr.write("}\n//End " + className + "\n\n\n\n " );
          intr.close();

          impl.write("}\n//End " + className + "\n\n\n\n " );
          impl.close();
        }


       return true;
  }//start()
```

```
    public static String ge   tendableName(ClassDoc cd) {
        /* Climb returns the name of the extended class of the ClassDoc Passed in
           If the superclass is a Swing Component, that is the extended class,
           If it is not a Swing component, the extended class is UnicastRemoteObject
         */
        ClassDoc parent = cd.superclass();

        if(!parent.isPublic()) return "UnicastRemoteObject";

        if (isRemoteComponent(parent)) {
            String className = "";
            String packageName = "edu.columbia.cs.cgui.rjfc";
            StringTokenizer st = new StringTokenizer(parent.toString(), ".");
            className = st.nextToken(); //discard java or javax
            className = st.nextToken();
            while (st.hasMoreTokens()) {
                packageName += "."+className; //ensures that the correct package is extended
                className = st.nextToken();
            }
            return (packageName+".R"+className);
        }
        return "UnicastRemoteObject";

    }//getExtendableName()

    public static String getReturnType(Type t) {
        ClassDoc cd = t.asClassDoc();
        if (isRemoteComponent(t)) {

            StringTokenizer st = new StringTokenizer(t.qualifiedTypeName(), ".");
            String className = st.nextToken();  //discard java or javax
            className = st.nextToken();
            String qualifiedPathName = "edu.columbia.cs.cgui.rjfc";
            while (st.hasMoreTokens()) {
                qualifiedPathName += "." + className;
                className = st.nextToken();
            }
            //System.out.println(qualifiedPathName);
            return (qualifiedPathName + ".R" + className /*+ p.type().dimension()*/);
        }
        String test = t.toString();
        if ((cd != null) && (cd.isInterface()) && (test.endsWith("Listener"))){
            StringTokenizer st = new StringTokenizer(t.qualifiedTypeName(), ".");
            String first, second, third;
            String token;
            if (st.countTokens() > 3) {
                first = st.nextToken();
                second = st.nextToken();
                third = st.nextToken();
                if (((first.equals("java"))&&(second.equals("awt"))||(first.equals("javax"))
&&(second.equals("swing"))) && (third.equals("event"))) {
                    token = st.nextToken();
                    while (st.hasMoreTokens()) {
                        token = st.nextToken();
                    }

                    String temp = "edu.columbia.cs.cgui.rjfc.event.R" + token;

                    return temp;
                }
            }
        }

        return t.qualifiedTypeName();
```

```
    public static String getComponentType(Parameter p) {
        return getReturnType(p.type()) + p.type().dimension() + " " + p.name();
    }

    public static String extractEventName(Parameter p,  String suff) {
        /*Returns the header for an EventListener
         * e.g. returns AWTEventCallBack for AWTEvent Listener
         * if not an Event Listener, returns a blank string
         */

        ClassDoc cd = p.type().asClassDoc();
        String test = p.type().toString();
        if ((cd != null) && (cd.isInterface()) && (test.endsWith("Listener"))) {
            StringTokenizer st = new StringTokenizer(p.type().qualifiedTypeName(), ".");
            String first, second, third, token;
            if (st.countTokens() > 3) {
                first = st.nextToken();
                second = st.nextToken();
                third = st.nextToken();
                if (((first.equals("java"))&&(second.equals("awt"))||(first.equals("javax"))
&&(second.equals("swing"))) && (third.equals("event"))) {
                    token = st.nextToken();
                    while (st.hasMoreTokens()) {
                        token = st.nextToken();
                    }
                    token = token.substring(0, token.length() - 8);
                    String temp = "new R" +token+ "CallBack("+p.name() + suff +")";

                    return temp;
                }
            }
        }
        return p.name() + suff;
    }
    public static boolean isRemoteComponent(Type t) {
        if (t.dimension().equals("")) {
            return isRemoteComponent(t.asClassDoc());
        }
        return false;
    }
    public static boolean isRemoteComponent(ClassDoc cd) {
        //ClassDoc cd = p.type().asClassDoc();

        if ((cd != null) && (!cd.isInterface()) && (cd.isPublic()) && (cd.containingClass()
== null) && (cd.dimension().equals(""))) {
            if (cd.superclass() != null) {
                if (cd.superclass().containingClass() != null) {
                    return false;
                }
            }
            StringTokenizer st = new StringTokenizer(cd.qualifiedTypeName(), ".");
            String first, second;
            if (st.countTokens() > 2) {
                first = st.nextToken();
                        second = st.nextToken();
                        if ((first.equals("javax"))&&(second.equals("swing"))) {
                            return true;
                        }
                        if ((first.equals("java"))&&(second.equals("awt"))) {
                            String third = st.nextToken();
                            if ((third.equals("Container"))||
                                (third.equals("Component"))||
                                (third.equals("Window"))||
```

```
                        (third.equals("Dialog"))||
                        (third.equals("Frame"))) {
                        return true;
                    }
                }
            }
        }
        return false;
    }

    public static void printConstructors(ClassDoc cd, String origVar, String origName) throw
s IOException {
        if (cd.isPublic()) {

            /* We will create a constructor that takes in a variable
               of the original Class
            */

            StringTokenizer st = new StringTokenizer(cd.toString(), ".");
            String className = "";

            while (st.hasMoreTokens()) {
                className = st.nextToken();
            }

            impl.write("\tpublic R" + className + "Impl(");
            impl.write(origName + " passedIn, java.util.Hashtable hash) throws RemoteExcepti
on {\n\t\t");
            impl.write("this.hash = hash;\n\t\t");
            impl.write(origVar + " = passedIn;\n\t}\n");

            ConstructorDoc[] cs = cd.constructors();

            /*
              CLASSES BorderFactory, SwingUtilities, KeyStroke and
              ToolTipManager do not have only private access
            */

            if (cs.length == 0) {
                //This happens if the class does not have a defined constructor
                //In this case, we just create an empty constructor
                impl.write("\tpublic ");
                //st = new StringTokenizer(cd.toString(), ".");
                //className = "";

                impl.write("R" + className +"Impl() throws RemoteException {\n\t\t");
                //impl.write(cd.isAbstract()?"":origVar + " = new " + cd.toString() + "(); "
);
                impl.write("\n\t}\n\n");
                return;
            }

            //boolean has0arg_ctor = false; //where ctor = constructor
            for (int j = 0; j < cs.length; j++) {
                if (!cs[j].isPublic())  continue;

                impl.write("\tpublic ");
                impl.write("R" + cs[j].toString()+"Impl");
                Parameter[] params = cs[j].parameters();
                ClassDoc[] exceps = cs[j].thrownExceptions();
                boolean hasexceps = (exceps.length > 0);

                if (params.length == 0) {
                    //has0arg_ctor = true;
                    impl.write("(java.util.Hashtable hash) throws RemoteException");
```

```
                    if (has   eps) {
                        for (int k=0; k < exceps.length; k++) {
                            impl.write(", " + exceps[k]);
                        }
                    }
                    impl.write("{ \n\t\t");
                    impl.write("this.hash = hash;\n\t\t");
                    impl.write((cd.isAbstract()?"":
                                origVar + " = new " + cs[j] +"();") +
                               "\n\t}\n");
                }
                else {
                    impl.write("(");
                    //IF parameter is in the swing class,
                    //replace with corresponding rjfc.swing class
                    impl.write(getComponentType(params[0])+VARSUFFIX+"0");
                    //System.out.println(getComponentType(params[0]));
                    for (int k = 1; k < params.length; k++) {
                        impl.write(", " + getComponentType(params[k])+VARSUFFIX+k);
                    }
                    impl.write(", java.util.Hashtable hash) throws RemoteException");
                    if (hasexceps) {
                        for (int k=0; k < exceps.length; k++) {
                            impl.write(", " + exceps[k]);
                        }
                    }
                    impl.write("{\n\t\t");
                    impl.write("this.hash = hash;\n\t\t");

                    if (!cd.isAbstract()){
                        impl.write(origVar +" = new " + cs[j] + "(");
                        //impl.write(isSwingComponent(params[0])?"(" + params[0].type().qual
ifiedTypeName() + ")":"");
                        impl.write(isRemoteComponent(params[0].type())?"("+params[0].type().
qualifiedTypeName()+")hash.get(":"");
                        impl.write(extractEventName(params[0],VARSUFFIX+"0"));
                        impl.write(isRemoteComponent(params[0].type())?")":"");
                        for (int k = 1; k < params.length; k++) {
                            impl.write(", ");
                            //impl.write(isSwingComponent(params[k])?"(" + params[k].type().
qualifiedTypeName() + ")":"");
                            impl.write(isRemoteComponent(params[k].type())?"("+params[k].typ
e().qualifiedTypeName()+")hash.get(":");
                            impl.write(extractEventName(params[k],VARSUFFIX+k));
                            impl.write(isRemoteComponent(params[k].type())?")":"");
                        }
                        impl.write(");");
                    }
                    impl.write("\n\t}\n");
                }
            }//big for

            impl.write("\tprotected ");
            impl.write("R" + className +"Impl() throws RemoteException {\n\t\t");
            impl.write("\n\t}\n\n");


        }
    }

    public static void printFields(ClassDoc cd, String origName) throws IOException{
        //System.out.println("Extracting Fields from class" + cd.toString());
          //I only want to output public classes when creating the Interface
          if (cd.isPublic()) {
```

```
            //Let's list all the fields first.
            FieldDoc[] fields = cd.fields();

            for (int j = 0; j < fields.length; j++) {
                //Again, I'm only concerned with public fields

                if (fields[j].isStatic()) {
                    if (fields[j].isPublic()) {
                        intr.write("\tpublic ");

                        intr.write("static ");

                        if (fields[j].isFinal()) {
                            intr.write("final ");
                        }

                        if(fields[j].isTransient()) {
                            intr.write("transient ");
                        }

                        if(fields[j].isVolatile()) {
                            intr.write("volatile ");
                        }

                        intr.write(fields[j].type() + " " + fields[j] + "= " +
                                origName + "." + fields[j] + ";\n");
                    }
                }
            }
            climbFields(cd, origName);
        }

    }//printFields()

    public static void printMethods(ClassDoc cd, String origVar, String origName) throws IOE
xception{
        //System.out.println("Extracting Methods from class" + cd.toString());

        String signature; //each method's signature will be put in the hash.

        //I only want to output public classes when creating the Interface
        if (cd.isPublic()) {

            //Now it's time to get the methods
            MethodDoc[] methods = cd.methods();

            for (int j = 0; j < methods.length; j++) {
                /*
                  Try to put the method's signature in the hash.
                  If it already exists, then skip it.
                  Note: the signature() method returns a String
                  that contains the types of the parameters of the method only,
                  and not the method's name.
                */
                signature = methods[j].name()+methods[j].signature();
                if(hash.contains(signature)) continue;
                else hash.add(signature);


                if (methods[j].isPublic()) {

                    //System.err.println(methods[j].name()+methods[j].signature());

                    /* Determines if the method will end with a return statement or not */
```

```
                String returnType = isRemoteComponent(methods[j].returnType())?getReturn
Type(methods[j].returnType()):methods[j].returnType().qualifiedTypeName();
                boolean returnsdiff = !returnType.equals((methods[j].returnType().qualif
iedTypeName()));
                returnType += methods[j].returnType().dimension();
                boolean isVoid = returnType.equals("void");
                intr.write("\tpublic " + returnType + " ");
                intr.write(methods[j].toString());
                impl.write("\tpublic ");
                impl.write(returnType + " ");
                impl.write(methods[j].toString());

                /*  Here i'm getting the paramaters for the method.  After that,
                    I'm determining if there are any paramaters at all.  If not,
                    then I do the first action in the if statement. ELSE, I list
                    the arguments one at a time.
                */

                Parameter[] params = methods[j].parameters();
                ClassDoc[] exceps = methods[j].thrownExceptions();
                boolean hasexceps = (exceps.length > 0);

                if (params.length == 0) {
                    intr.write("() throws RemoteException");
                    impl.write("() throws RemoteException");
                    if (hasexceps) {
                        for (int k=0; k < exceps.length; k++) {
                            intr.write(", " + exceps[k]);
                            impl.write(", " + exceps[k]);
                        }
                    }
                    intr.write(";\n");
                    impl.write("{ \n\t\t");

                    impl.write((isVoid?"":returnType + " temp = " +
                                (returnsdiff?" new " + returnType + "Impl(":"))
                               + (methods[j].isStatic()?origName:origVar)+
                               "." + methods[j] + "()" +
                               (returnsdiff?", hash)":"")+
                               ";\n");

                }
                else {
                    intr.write("("+ getComponentType(params[0])+VARSUFFIX+"0");
                    impl.write("("+ getComponentType(params[0])+VARSUFFIX+"0");
                    for (int k = 1; k < params.length; k++) {
                        intr.write(", "+ getComponentType(params[k])+VARSUFFIX+k);
                        impl.write(", "+ getComponentType(params[k])+VARSUFFIX+k);
                    }
                    intr.write(") throws RemoteException");
                    impl.write(") throws RemoteException");
                    if (hasexceps) {
                        for (int k=0; k < exceps.length; k++) {
                            intr.write(", " + exceps[k]);
                            impl.write(", " + exceps[k]);
                        }
                    }
                    intr.write(";\n");
                    impl.write("{ \n\t\t");
                    impl.write((isVoid?"":returnType + " temp = " +
                                (returnsdiff?" new " + returnType + "Impl(":"")) +
                               (methods[j].isStatic()?origName:origVar) + "." + methods[
j] + "(");
                    //impl.write(isSwingComponent(params[0])?"(" + params[0].type().qual
```

```
ifiedTypeName() + ")":"");
                        impl.write(isRemoteComponent(params[0].type())?"("+params[0].type().
qualifiedTypeName()+params[0].type().dimension()+")hash.get(":"");
                        impl.write(extractEventName(params[0],VARSUFFIX+"0"));
                        impl.write(isRemoteComponent(params[0].type())?")":"");
                        for (int k = 1; k < params.length; k++) {
                            impl.write(", ");
                            //impl.write(isSwingComponent(params[k])?"(" + params[k].type().
qualifiedTypeName() + ")":"");
                            impl.write(isRemoteComponent(params[k].type())?"("+params[k].typ
e().qualifiedTypeName()+params[k].type().dimension()+")hash.get(":"");
                            impl.write(extractEventName(params[k],VARSUFFIX+k));
                            impl.write(isRemoteComponent(params[k].type())?")":"");
                        }
                        impl.write((returnsdiff?"), hash":"") + ");\n");
                    }
                    impl.write(returnsdiff?"\t\thash.put(temp, temp.getRJFCWrappedData());\n
\t\ttemp.setHash(hash);":"");
                    impl.write(isVoid?"":"\t\treturn temp;\n");
                    impl.write("\t}\n");
                }
            }
            climbMethods(cd, origVar, origName);
        }
    }//printMethods()

    public static void climbFields(ClassDoc cd, String origName) throws IOException {
        //Check to see if parent is a swing component
        if (getExtendableName(cd).equals("UnicastRemoteObject")) {
            ClassDoc cds = cd.superclass();
            if (!cds.toString().equals("java.lang.Object")){
                printFields(cds, origName);
            }
        }
        return;
    }//climbFields()

    public static void climbMethods(ClassDoc cd, String origVar, String origName) throws IOE
xception {
        //TESTING PURPOSES
        test = true;

        //Check to see if parent is a swing component
        if (getExtendableName(cd).equals("UnicastRemoteObject")) {
            ClassDoc cds = cd.superclass();
            if (!cds.toString().equals("java.lang.Object")){
                printMethods(cds, origVar, origName);
            }
        }
        return;
    }//climbMethods()

}//class
```

```
import javax.swing.*;
import java.net.InetAddress;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
//import java.awt.*;
import java.rmi.*;
import java.rmi.server.*;
import edu.columbia.cs.cgui.rjfc.swing.RJFrameImpl;
import edu.columbia.cs.cgui.rjfc.swing.RJFrame;
import edu.columbia.cs.cgui.rjfc.RJFCFactoryImpl;
import edu.columbia.cs.cgui.rjfc.RJFCFactory;
import edu.columbia.cs.cgui.rjfc.Server;


public class Viewer extends JFrame
{
    private RJFCFactory fact;
    private RJFrame display;
    private JTextField locationField;
    private JFrame clientComponents;
    //private JFrame serverComponents;
    private JButton connectButton;
    private JButton disconnectButton;
    private JTextArea statusBar;
    private JScrollPane statusScroll;
    private JMenuItem exitMenuItem;
    private Server theServer;
    private static final String DEFAULT_URL = "okeeffe/RJFCTest";

    public Viewer(){

        setSize(500, 500);

        //Container contentPane = getContentPane();

        clientComponents = new JFrame("RJFC Client");
        clientComponents.getContentPane().setLayout(new java.awt.GridLayout(3,1));

        statusScroll = new JScrollPane();
        JMenuBar menuBar = new JMenuBar();
        JMenu fileMenu = new JMenu("File");
        exitMenuItem = new JMenuItem("Exit");
        exitMenuItem.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent evt) {
                    System.exit(0);
                }
            });

        fileMenu.add(exitMenuItem);
        menuBar.add(fileMenu);
        //setJMenuBar(menuBar);

        connectButton = new JButton("Connect to:");
        connectButton.setToolTipText("Specify address in field");
        connectButton.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent evt) {
                    connectToServer(locationField.getText());
                }
            });

        disconnectButton = new JButton("Disconnect");
        disconnectButton.setToolTipText("Disconnect from server");
        disconnectButton.addActionListener(new java.awt.event.ActionListener() {
```

```
            public voi  ctionPerformed(java.awt.event.Act:  Event evt) {
                disconnect();
            }
        });
    locationField = new JTextField(DEFAULT_URL, 20);
    locationField.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                connectButton.doClick();
            }
        });

    clientComponents.setJMenuBar(menuBar);
    JPanel connectPanel = new JPanel();

    //clientComponents.getContentPane().add(connectButton);
    //clientComponents.getContentPane().add(locationField);
    connectPanel.add(locationField);
    connectPanel.add(connectButton);

    clientComponents.getContentPane().add(connectPanel);
    clientComponents.getContentPane().add(disconnectButton);
    //serverComponents = new JPanel();
    statusBar = new JTextArea(" ");

    statusBar.setEditable(false);
    statusBar.setLineWrap(true);

    statusScroll.setViewportView(statusBar);
    statusScroll.setPreferredSize(new java.awt.Dimension(240, 50));

    //contentPane.add("North", clientComponents);

    clientComponents.getContentPane().add(statusScroll);
    //contentPane.add(serverComponents);
    locationField.requestFocus();

    setStatus("Started the Viewer");
    clientComponents.pack();
    clientComponents.show();
    clientComponents.addWindowListener(new WindowAdapter() {
            public void windowClosing( WindowEvent e) {
                System.exit(0);
            }
        });
    getRootPane().setDefaultButton(connectButton);
}

public static void main(String args[])
{

    Viewer viewer = new Viewer();
    //viewer.show();
    viewer.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                //e.consume();
                exitForm((Viewer) e.getWindow());
            }
        });
}

private void setStatus(String statusmessage) {
    statusBar.append(statusmessage + '\n');
}

private void connectToServer (String serverName) {
```

```
        if (getServer(serv   .ame)) {
            show();
        }
    }



    private static void exitForm(Viewer v) {
        v.disconnect();
    }


    public void setServer(Server theServer) throws RemoteException {
        java.util.Hashtable hash = new java.util.Hashtable();
        //OurHashtable hash = new OurHashtable();
        fact = new RJFCFactoryImpl(hash);
        //display = fact.getRJPanel();

        //javax.swing.JPanel jpanel = new javax.swing.JPanel();
        //display = new RJPanelImpl(jpanel);
        display = new RJFrameImpl(this,hash);
        hash.put(display,display.getRJFCWrappedData());

        //      System.out.println("jpanel clean: "+System.identityHashCode(jpanel));
        System.out.println("display hash: "+System.identityHashCode(hash.get(display)));
        System.out.println("display getdata: "+System.identityHashCode(display.getRJFCWrappe
dData()));



        //display.setData(this);
        //System.out.println("display before: "+System.identityHashCode(display.getRJFCWrapp
edData()));
        //System.out.println("display before from hash: "+System.identityHashCode(hash.get(d
isplay)));
        this.theServer = theServer;
        System.err.println("Found a server " + theServer);

        try {
            theServer.registerDisplay(display, fact, InetAddress.getLocalHost());
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        setStatus("Connected to the server.\n");
    }

    public void hide() {
        reset();
        super.hide();
    }

    private void disconnect() {

        if (theServer != null) {
            try {
                System.out.println("display on disconnect: "+System.identityHashCode(display
));
                theServer.disconnectDisplay(display);
            } catch (RemoteException e) {
                System.out.println(e.getMessage());
            }
            setStatus("Disconnected from Server");
            //serverComponents = new JPanel();
```

```
        }
        reset();
    }

    private void reset() {
        theServer = null;
        try {
            display.getContentPane().removeAll();
        }catch (RemoteException e) {
            e.printStackTrace();
        }
            //hide();
    }

    public Server getServer() {
        return theServer;
    }

    /** attempt to make contact with the server... */
    private boolean getServer(String serverName) {
        try {
            String rmiURL = "rmi://"+ locationField.getText();
            System.out.println("Connecting to "+rmiURL);
            setStatus("Attempting to Connect toServer " + rmiURL);

            System.out.println("looking up "+rmiURL);
            Server rmiServer = (Server)Naming.lookup(rmiURL);
            System.out.println("found "+rmiURL);
            setServer(rmiServer);
            return true;
            //setVisible(false);
            //dispose();
        } catch (Exception e) {
           setStatus("Invalid Server");
           System.out.println(e);
           e.printStackTrace();
        }
       return false;
    }
}

class OurHashtable extends java.util.Hashtable {

    public Object put(Object k, Object v) {
        System.out.println();
        if(containsKey(k)) System.out.println("*** key already exists ***");
        System.out.println("put:  key = "+k.getClass()+", val = "+v.getClass());
        System.out.println();
        return super.put(k,v);

    }

    public Object get(Object k) {
        System.out.println();
        if(!containsKey(k)) System.out.println("** key does not exist ***");
        System.out.println("get: key = "+k.getClass());
        System.out.println();
        return super.get(k);
    }

    public Object remove(Object k) {
        System.out.println();
        if(!containsKey(k)) System.out.println("** key does not exist ***");
        System.out.println("remove: key = "+k.getClass());
        System.out.println();
```

```
        return super.remov.   );
    }
}
```

```
import java.net.*;
import java.rmi.RemoteException;
import java.rmi.registry.*;
import edu.columbia.cs.cgui.rjfc.swing.*;
import edu.columbia.cs.cgui.rjfc.event.*;
import edu.columbia.cs.cgui.rjfc.RJFCFactory;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.*;
import java.awt.Component;
import java.net.*;
import java.awt.event.*;
import javax.swing.ImageIcon;


public class NotepadServer extends UnicastRemoteObject implements Server{

    //Fields:
    /* should create this in a thread, but we're lazy */

    //DOCSender for Dynamic Object Caching
    //DOCSender docSender;


    RJWindow aboutWindow;
    //RJLabel aboutText;

    RJDialog findDialog;
    RJTextField findTextField;

    RJDialog openDialog;
    RJTextField openTextField;

    RJDialog saveDialog;
    RJTextField saveTextField;

    InetAddress clientAddress;

    RJTextArea textArea;
    RJFrame display;
    RJFCFactory f;
    RJFrame p;
    boolean isFileModified;
    RJTextField statusBar;
    String currentOpenFile=null;
    String currentFile="Untitled";
    String currentFilePath=System.getProperty("user.home",".");
    ImageIcon copy = new ImageIcon("./images/copy.gif");
    ImageIcon paste = new ImageIcon("./images/paste.gif");
    ImageIcon cut = new ImageIcon("./images/cut.gif");
    ImageIcon open = new ImageIcon("./images/open.gif");
    ImageIcon save = new ImageIcon("./images/save.gif");
    ImageIcon newDoc  = new ImageIcon("./images/new.gif");
    final String VERSION = "1.0";

    //String clipBoard;
    //ServerConsole console;

    /** Constructs NotepadServer object and exports it on default port.
     */
    public NotepadServer() throws RemoteException {
        super();

    }
```

```
    /** Constructs NotepadServer object and exports it on specif. d port.
     * @param port The port for exporting
     */
    public NotepadServer(int port) throws RemoteException {
        super(port);

    }

    /** Main method.
     */
    public static void main(String[] args) {


        System.setSecurityManager(new RMISecurityManager());

        String rmiName = "NotepadServer";

        try {
            NotepadServer obj = new NotepadServer ();
            registerToRegistry(rmiName, obj, true);
            //console.show();
            //console.message("Server ready, rmiName is " + rmiName);
            System.out.println("Server ready, rmiName = " + rmiName);

        } catch (RemoteException ex) {
            ex.printStackTrace();
        } catch (MalformedURLException ex) {
            ex.printStackTrace();
        }

    }

    /** Register NotepadServer with RMI registry
     *
     */
    public static void registerToRegistry(String name, Remote obj, boolean create) throws Re
moteException, MalformedURLException{

        if (name == null) throw new IllegalArgumentException("registration name can not be n
ull");

        try {
            Naming.rebind(name, obj);
        } catch (RemoteException ex){
            if (create) {
                Registry r = LocateRegistry.createRegistry(Registry.REGISTRY_PORT);
                r.rebind(name, obj);
            } else throw ex;
        }
    }


    /**
     * This method should be called by clients when they first initialize.  It
     * should pass the display hook to the server so that the server can
     * use RMI to add components onto the display panel
     */

    public void registerDisplay(RJFrame display, RJFCFactory f,InetAddress clientAddr) throw
s RemoteException {
        this.clientAddress = clientAddr;
        this.f = f;
        System.out.println("A Display has registered\n");
        this.display = display;
        textArea = f.getRJTextArea(20,20);
```

```
        RJScrollPane pane    ..getRJScrollPane();
        pane.setViewportView(textArea);
        statusBar = f.getRJTextField();

        statusBar.setEditable(false);

        System.out.println();
        //aboutWindow = f.getRJWindow();
        //aboutWindow.show();
        edu.columbia.cs.cgui.rjfc.awt.RContainer rc;

        //docSender = new DOCSender(clientAddr);

        try {
            initializeSaveDialog();
            initializeOpenDialog();
            initializeFindDialog();
            initializeAboutDialog();
            //textArea.addKeyListener(new TextAreaKeyListener());
            rc = this.display.getContentPane();
            rc.setLayout(new java.awt.BorderLayout());
            rc.add(statusBar, java.awt.BorderLayout.SOUTH);
            rc.add(createMenu(), java.awt.BorderLayout.NORTH);

            rc.add(pane);

            display.show();
        } catch (RemoteException e) {
            System.out.println("prob Adding Component" + e);
        }

    }


    //this method should only be called by the constructor
    //to initialize the find dialog box
    private void initializeSaveDialog() throws RemoteException {
        saveDialog = f.getRJDialog(display,"Save",false);
        saveTextField = f.getRJTextField(20);

        edu.columbia.cs.cgui.rjfc.awt.RContainer rc;
        rc = saveDialog.getContentPane();

        rc.setLayout(new java.awt.FlowLayout());

        RJLabel label = f.getRJLabel("File to Save: ");

        RJButton buttonOK = f.getRJButton("OK");
        RJButton buttonCancel = f.getRJButton("Cancel");

        rc.add(label);
        rc.add(saveTextField);
        rc.add(buttonOK);
        rc.add(buttonCancel);

        SaveListener listener = new SaveListener();
        saveTextField.addActionListener(listener);
        buttonCancel.addActionListener(listener);
        buttonOK.addActionListener(listener);

        saveDialog.setSize(new java.awt.Dimension(400,90));
    }


    //this method should only be called by the constructor
```

```
//to initialize the fi. · dialog box
private void initializeOpenDialog() throws RemoteException {
    openDialog = f.getRJDialog(display,"Open",false);
    openTextField = f.getRJTextField(20);

    edu.columbia.cs.cgui.rjfc.awt.RContainer rc;
    rc = openDialog.getContentPane();

    rc.setLayout(new java.awt.FlowLayout());

    RJLabel label = f.getRJLabel("File to Open: ");

    RJButton buttonOK = f.getRJButton("OK");
    RJButton buttonCancel = f.getRJButton("Cancel");

    rc.add(label);
    rc.add(openTextField);
    rc.add(buttonOK);
    rc.add(buttonCancel);

    OpenListener listener = new OpenListener();
    openTextField.addActionListener(listener);
    buttonCancel.addActionListener(listener);
    buttonOK.addActionListener(listener);

    openDialog.setSize(new java.awt.Dimension(400,90));
}


//this method should only be called by the constructor
//to initialize the find dialog box
private void initializeFindDialog() throws RemoteException {
    findDialog = f.getRJDialog(display,"Find",false);
    findTextField = f.getRJTextField(20);

    edu.columbia.cs.cgui.rjfc.awt.RContainer rc;
    rc = findDialog.getContentPane();

    rc.setLayout(new java.awt.FlowLayout());

    RJLabel label = f.getRJLabel("Search String: ");

    RJButton buttonOK = f.getRJButton("OK");
    RJButton buttonCancel = f.getRJButton("Cancel");

    rc.add(label);
    rc.add(findTextField);
    rc.add(buttonOK);
    rc.add(buttonCancel);

    FindListener listener = new FindListener();
    findTextField.addActionListener(listener);
    buttonCancel.addActionListener(listener);
    buttonOK.addActionListener(listener);

    findDialog.setSize(new java.awt.Dimension(400,90));

}

private void showErrorMessage(String msg) {
    System.out.println("Error: "+msg);
    try {
        RJOptionPane pane = f.getRJOptionPane(msg,RJOptionPane.ERROR_MESSAGE);
        RJDialog errorDialog = pane.createDialog(display,"Error");
        errorDialog.setModal(false);
```

```
            errorDialog.sh  );
        }
        catch(RemoteException e) {
            e.printStackTrace();
        }
    }


    private void find(String s) throws RemoteException {
        System.out.println("Request to find: "+s);
        int start, end;

        String doc = textArea.getSelectedText();
        System.out.println("selected text: "+doc);

        start = textArea.getSelectionStart();
        end = textArea.getSelectionEnd();

        if(start == end) {
            doc = textArea.getText();
            start = 0;
            end = doc.length();
        }

        System.out.println("start: "+ start + "   end: "+end);


        for(int i=0;i<end-start;i++) {
            if(doc.startsWith(s,i)) {
                textArea.setSelectionStart(i+start);
                textArea.setSelectionEnd(i+start+s.length());
                return;
            }
            System.out.println();
        }
        showErrorMessage("No occurence of "+s+" found.");


    }

    private void open(String inFile) {
        System.out.println("Request to open: "+inFile);
        String file = "";
        try {
            java.io.FileReader fr = new java.io.FileReader (inFile);
            java.io.BufferedReader input = new java.io.BufferedReader(fr);
            StringBuffer line =  new StringBuffer();
            while(input.ready()) {
                line.append(input.readLine());
                line.append("\n");
            }
            textArea.setText(line.toString());
        }
        catch (java.io.FileNotFoundException e) {
            showErrorMessage("Sorry, "+inFile+" was not found.");
        }
        catch (java.io.IOException e) {
            showErrorMessage(e.getMessage());
        }
        catch (java.lang.NumberFormatException e) {
            showErrorMessage(e.getMessage());
        }
```

```
    private void save(String outFile) {
        System.out.println("Request to save into: "+outFile);
        try {
            java.io.PrintWriter output = new java.io.PrintWriter(new java.io.FileOutputStrea
m(outFile));
            output.print (textArea.getText());
            output.close();

        }
        catch (java.io.FileNotFoundException e) {
            showErrorMessage(e.getMessage());
        }
        catch (java.io.IOException e) {
            showErrorMessage(e.getMessage());
        }
        catch (java.lang.NumberFormatException e) {
            showErrorMessage(e.getMessage());
        }

    }


    //this method should only be called by the constructor
    //to initialize the find dialog box
    private void initializeAboutDialog() throws RemoteException {


        String description  = "Notepad Server v" + VERSION + " on " + System.getProperty("os
.name",".");

        System.out.println(description);
        aboutWindow = f.getRJWindow();
        RJLabel aboutText = f.getRJLabel(description);
        aboutWindow.setSize(200,200);
        edu.columbia.cs.cgui.rjfc.awt.RContainer rc = aboutWindow.getContentPane();
        RJLabel rjl = f.getRJLabel(description);
        rc.add(rjl);
        aboutWindow.addMouseListener(new AboutWindowListener());


    }

    private RJMenuBar createMenu() throws RemoteException {
        //
        //RJFrame p = f.getRJFrame();
        //p = f.getRJFrame();

        //add actionlisteners later
        RJMenuBar menuBar = f.getRJMenuBar();
        menuBar.setBorderPainted(true);
        //menuBar.setFloatable(false);
        RJMenu fileMenu = f.getRJMenu("File");
        RJMenu editMenu = f.getRJMenu("Edit");
        RJMenu helpMenu = f.getRJMenu("Help");

        //      System.out.println ("1");
        System.out.println ("2");
        RJMenuItem newMenuItem = f.getRJMenuItem("New", newDoc);
        newMenuItem.addActionListener(new NewListener());
        RJMenuItem openMenuItem = f.getRJMenuItem("Open", open);
        openMenuItem.addActionListener(new OpenDialogListener());
        RJMenuItem saveMenuItem = f.getRJMenuItem("Save", save);
```

```
        saveMenuItem.addActionListener(new SaveDialogListener());
        RJMenuItem exitMenuItem = f.getRJMenuItem("Exit");
        exitMenuItem.addActionListener(new ExitListener());
        RJMenuItem copyMenuItem = f.getRJMenuItem("Copy", copy);
        copyMenuItem.addActionListener(new ClipboardListener());
        RJMenuItem pasteMenuItem = f.getRJMenuItem("Paste", paste);
        pasteMenuItem.addActionListener(new ClipboardListener());
        RJMenuItem cutMenuItem = f.getRJMenuItem("Cut", cut);
        cutMenuItem.addActionListener(new ClipboardListener());

        RJMenuItem findMenuItem = f.getRJMenuItem("Find");
        findMenuItem.addActionListener(new FindDialogListener());
        RJMenuItem aboutMenuItem = f.getRJMenuItem("About");
        aboutMenuItem.addActionListener(new AboutListener());
        System.out.println ("3");
        fileMenu.add(newMenuItem);
        fileMenu.add(openMenuItem);
        fileMenu.add(saveMenuItem);

        System.out.println ("4");
        fileMenu.addSeparator();
        fileMenu.add(exitMenuItem);

        System.out.println ("5");
        editMenu.add(copyMenuItem);
        editMenu.add(pasteMenuItem);
        editMenu.add(cutMenuItem);
        editMenu.addSeparator();
        editMenu.add(findMenuItem);

        helpMenu.add(aboutMenuItem);

        menuBar.add(fileMenu);
        menuBar.add(editMenu);
        menuBar.add(helpMenu);
        System.out.println ("6");
        //RJLabel l = f.getRJLabel("test");
        //System.out.println ("7");
        //p.addRComponent(menuBar);
        System.out.println ("8");
        return menuBar;
    }

    class NewListener extends RActionAdapter {
        public NewListener() throws RemoteException{
            super();
        }

        public void actionPerformed(ActionEvent e) throws RemoteException {
            textArea.setText("");
        }
    }

    class AboutListener extends RActionAdapter {
        public AboutListener() throws RemoteException{
            super();
        }

        public void actionPerformed(ActionEvent e) throws RemoteException {
            System.out.println("about called");
            aboutWindow.show();
        }
    }

    class FindDialogListener extends RActionAdapter {
```

```
    public FindDialogListener() throws RemoteException {
        super();
    }

    public void actionPerformed(ActionEvent e) throws RemoteException {
        System.out.println("find dialog called");
        findDialog.show();
    }
}

class FindListener extends RActionAdapter {


    public FindListener() throws RemoteException {
        super();
    }

    public void actionPerformed(ActionEvent e) throws RemoteException {
        System.out.println("find called");
        if(!e.getActionCommand().equals("Cancel"))
            find(findTextField.getText());
        findDialog.hide();
    }


}


class SaveDialogListener extends RActionAdapter {
    public SaveDialogListener() throws RemoteException {
        super();
    }
    public void actionPerformed(ActionEvent e) throws RemoteException {
        System.out.println("save dialog called");
        saveDialog.show();
    }
}

class SaveListener extends RActionAdapter {
    public SaveListener() throws RemoteException {
        super();
    }
    public void actionPerformed(ActionEvent e) throws RemoteException {
        System.out.println("save called");
        if(!e.getActionCommand().equals("Cancel"))
            save(saveTextField.getText());
        saveDialog.hide();
    }
}


class AboutWindowListener extends RMouseAdapter {
    public AboutWindowListener() throws RemoteException {
        super();
    }

    public void mouseClicked(MouseEvent e) throws RemoteException {
        aboutWindow.hide();
    }
}



class OpenDialogListener extends RActionAdapter {
    public OpenDialogListener() throws RemoteException {
```

```
        super();
    }

    public void actionPerformed(ActionEvent e) throws RemoteException {
        System.out.println("open dialog called");
        openDialog.show();
        System.out.println();
    }
}

class OpenListener extends RActionAdapter {
    public OpenListener() throws RemoteException {
        super();
    }

    public void actionPerformed(ActionEvent e) throws RemoteException {
        System.out.println("open called");
        if(!e.getActionCommand().equals("Cancel"))
            open(openTextField.getText());
        openDialog.hide();
    }
}


class TextAreaKeyListener extends RKeyAdapter {
    public TextAreaKeyListener() throws RemoteException {
        super();
    }

    public void keyTyped(KeyEvent e) throws RemoteException {
        System.out.print("Keytyped: "+e.getKeyChar());
    }

}


class ClipboardListener extends RActionAdapter {
    public ClipboardListener() throws RemoteException{
        super();
    }

    public void actionPerformed(ActionEvent e) throws RemoteException {
        if (e.getActionCommand().equals("Cut")) {
            textArea.cut();
            //clipBoard = textArea.getSelectedText();
        }
        else if (e.getActionCommand().equals("Copy")) {
            //clipBoard = textArea.getSelectedText();
            textArea.copy();
        }
        else if (e.getActionCommand().equals("Paste")) {
            textArea.paste();
            //textArea.replaceSelection(clipBoard);
        }
        else {
            System.out.println("Invalid Clipboard call");
        }
    }
}


class ExitListener extends RActionAdapter {
    public ExitListener() throws RemoteException{
        super();
    }
```

```
    public void actionPerformed(ActionEvent e) throws RemoteException {
        disconnectDisplay(display);
        //display.serverHide();
    }
}

public void disconnectDisplay(RJFrame display) throws RemoteException {
    display.hide();
    this.display = null;
    //console.message("The Display has disconnected\n");
    //
    System.out.println("Display disconnected");
}
```

```
package edu.columbia.cs.cgui.rjfc;

import java.net.*;
import java.rmi.RemoteException;
import java.rmi.registry.*;
//import edu.columbia.cs.cgui.rjfc.RJFCFactory;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.*;
import edu.columbia.cs.cgui.rjfc.swing.RJFrame;
//import edu.columbia.cs.cgui.rjfc.RJFCFactory;

public class RJFCServer extends UnicastRemoteObject implements Server{

    //Fields:
    /* should create this in a thread, but we're lazy */

    //DOCSender for Dynamic Object Caching
    //DOCSender docSender;
    InetAddress clientAddress;

    RJFrame display;
    RJFCFactory f;

    RJFCApplication app;
    String rmiName;

    /** Constructs RJFCServer object and exports it on default port.
     */
    public RJFCServer(String s, RJFCApplication ra) throws RemoteException {
        super();
        rmiName = s;
        app = ra;
        post(s);
    }

    public RJFrame getDisplay() {
        return display;
    }

    public InetAddress getClientAddress() {
        return clientAddress;
    }

    public RJFCFactory getFactory() {
        return f;
    }

    /** Constructs RJFCServer object and exports it on specified port.
     * @param port The port for exporting
     */
    public RJFCServer(int port, String s, RJFCApplication ra) throws RemoteException {
        super(port);
        app = ra;
        rmiName = s;
        post(s);
    }

    /** Main method.
     */
    private void post(String s) {

        System.setSecurityManager(new RMISecurityManager());

        String rmiName = s;
```

```
        try {
            //RJFCServer obj = new RJFCServer ();
            registerToRegistry(rmiName, this, true);
            //console.show();
            //console.message("Server ready, rmiName is " + rmiName);
            System.out.println("Server ready, rmiName = " + rmiName);

        } catch (RemoteException ex) {
            ex.printStackTrace();
        } catch (MalformedURLException ex) {
            ex.printStackTrace();
        }

    }

    /** Register RJFCServer with RMI registry
     *
     */
    private static void registerToRegistry(String name, Remote obj, boolean create) throws R
emoteException, MalformedURLException{

        if (name == null) throw new IllegalArgumentException("registration name can not be n
ull");

        try {
            Naming.rebind(name, obj);
        } catch (RemoteException ex){
            if (create) {
                Registry r = LocateRegistry.createRegistry(Registry.REGISTRY_PORT);
                r.rebind(name, obj);
            } else throw ex;
        }
    }


    /**
     * This method should be called by clients when they first initialize.  It
     * should pass the display hook to the server so that the server can
     * use RMI to add components onto the display panel
     */

    public void registerDisplay(RJFrame display, RJFCFactory f,InetAddress clientAddr) throw
s RemoteException {
        this.clientAddress = clientAddr;
        this.f = f;
        System.out.println("A Display has registered\n");
        this.display = display;

        app.begin();
    }

    public void disconnectDisplay(RJFrame display) throws RemoteException {
        display.hide();
        this.display = null;

        //console.message("The Display has disconnected\n");
        //
        app.end();
        System.out.println("Display disconnected");

    }

}
```

```
package edu.columbia.cs.cgui.rjfc;

import java.rmi.RemoteException;

public interface RJFCFactory extends java.rmi.Remote {

        public RJFCBorderFactory getBorderFactory() throws RemoteException;
        public RJFCBorderFactory border() throws RemoteException;
        public RJFCColorChooserFactory getColorChooserFactory() throws RemoteException;
        public RJFCColorChooserFactory colorchooser() throws RemoteException;
        public RJFCFileChooserFactory getFileChooserFactory() throws RemoteException;
        public RJFCFileChooserFactory filechooser() throws RemoteException;
        public RJFCPlafFactory getPlafFactory() throws RemoteException;
        public RJFCPlafFactory plaf() throws RemoteException;
        public RJFCTableFactory getTableFactory() throws RemoteException;
        public RJFCTableFactory table() throws RemoteException;
        public RJFCTextFactory getTextFactory() throws RemoteException;
        public RJFCTextFactory text() throws RemoteException;
        public RJFCTreeFactory getTreeFactory() throws RemoteException;
        public RJFCTreeFactory tree() throws RemoteException;
        public RJFCUndoFactory getUndoFactory() throws RemoteException;
        public RJFCUndoFactory undo() throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RActionMap getRActionMap() throws RemoteExcep
tion ;
        public edu.columbia.cs.cgui.rjfc.swing.RBox getRBox(int axisvar0) throws RemoteExcep
tion;
        public edu.columbia.cs.cgui.rjfc.swing.RBoxLayout getRBoxLayout(edu.columbia.cs.cgui
.rjfc.awt.RContainer targetvar0, int axisvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RButtonGroup getRButtonGroup() throws RemoteE
xception ;
        public edu.columbia.cs.cgui.rjfc.swing.RCellRendererPane getRCellRendererPane() thro
ws RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RComponentInputMap getRComponentInputMap(edu.
columbia.cs.cgui.rjfc.swing.RJComponent componentvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RDebugGraphics getRDebugGraphics() throws Rem
oteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RDebugGraphics getRDebugGraphics(java.awt.Gra
phics graphicsvar0, edu.columbia.cs.cgui.rjfc.swing.RJComponent componentvar1) throws Remote
Exception;
        public edu.columbia.cs.cgui.rjfc.swing.RDebugGraphics getRDebugGraphics(java.awt.Gra
phics graphicsvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RDefaultBoundedRangeModel getRDefaultBoundedR
angeModel() throws RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RDefaultBoundedRangeModel getRDefaultBoundedR
angeModel(int valuevar0, int extentvar1, int minvar2, int maxvar3) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RDefaultButtonModel getRDefaultButtonModel()
throws RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RDefaultCellEditor getRDefaultCellEditor(edu.
columbia.cs.cgui.rjfc.swing.RJTextField textFieldvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RDefaultCellEditor getRDefaultCellEditor(edu.
columbia.cs.cgui.rjfc.swing.RJCheckBox checkBoxvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RDefaultCellEditor getRDefaultCellEditor(edu.
columbia.cs.cgui.rjfc.swing.RJComboBox comboBoxvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RDefaultComboBoxModel getRDefaultComboBoxMode
l() throws RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RDefaultComboBoxModel getRDefaultComboBoxMode
l(java.lang.Object[] itemsvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RDefaultComboBoxModel getRDefaultComboBoxMode
l(java.util.Vector vvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RDefaultDesktopManager getRDefaultDesktopMana
ger() throws RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RDefaultFocusManager getRDefaultFocusManager(
) throws RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RDefaultListCellRenderer getRDefaultListCellR
enderer() throws RemoteException ;
```

```
        public edu.columbia.cs.cgui.rjfc.swing.RDefaultListMod  getRDefaultListModel() thro
ws RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RDefaultListSelectionModel getRDefaultListSel
ectionModel() throws RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RDefaultSingleSelectionModel getRDefaultSingl
eSelectionModel() throws RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RGrayFilter getRGrayFilter(boolean bvar0, int
 pvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RImageIcon getRImageIcon(java.lang.String fil
enamevar0, java.lang.String descriptionvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RImageIcon getRImageIcon(java.lang.String fil
enamevar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RImageIcon getRImageIcon(java.net.URL locatio
nvar0, java.lang.String descriptionvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RImageIcon getRImageIcon(java.net.URL locatio
nvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RImageIcon getRImageIcon(java.awt.Image image
var0, java.lang.String descriptionvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RImageIcon getRImageIcon(java.awt.Image image
var0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RImageIcon getRImageIcon(byte[] imageDatavar0
, java.lang.String descriptionvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RImageIcon getRImageIcon(byte[] imageDatavar0
) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RImageIcon getRImageIcon() throws RemoteExcep
tion ;
        public edu.columbia.cs.cgui.rjfc.swing.RInputMap getRInputMap() throws RemoteExcepti
on ;
        public edu.columbia.cs.cgui.rjfc.swing.RJApplet getRJApplet() throws RemoteException
 ;
        public edu.columbia.cs.cgui.rjfc.swing.RJButton getRJButton() throws RemoteException
 ;
        public edu.columbia.cs.cgui.rjfc.swing.RJButton getRJButton(javax.swing.Icon iconvar
0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJButton getRJButton(java.lang.String textvar
0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJButton getRJButton(javax.swing.Action avar0
) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJButton getRJButton(java.lang.String textvar
0, javax.swing.Icon iconvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJCheckBox getRJCheckBox() throws RemoteExcep
tion ;
        public edu.columbia.cs.cgui.rjfc.swing.RJCheckBox getRJCheckBox(javax.swing.Icon ico
nvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJCheckBox getRJCheckBox(javax.swing.Icon ico
nvar0, boolean selectedvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJCheckBox getRJCheckBox(java.lang.String tex
tvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJCheckBox getRJCheckBox(javax.swing.Action a
var0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJCheckBox getRJCheckBox(java.lang.String tex
tvar0, boolean selectedvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJCheckBox getRJCheckBox(java.lang.String tex
tvar0, javax.swing.Icon iconvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJCheckBox getRJCheckBox(java.lang.String tex
tvar0, javax.swing.Icon iconvar1, boolean selectedvar2) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJCheckBoxMenuItem getRJCheckBoxMenuItem() th
rows RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RJCheckBoxMenuItem getRJCheckBoxMenuItem(java
x.swing.Icon iconvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJCheckBoxMenuItem getRJCheckBoxMenuItem(java
.lang.String textvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJCheckBoxMenuItem getRJCheckBoxMenuItem(java
x.swing.Action avar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJCheckBoxMenuItem getRJCheckBoxMenuItem(java
```

```
.lang.String textvar0, javax.swing.Icon iconvar1) throws Remot    eption;
        public edu.columbia.cs.cgui.rjfc.swing.RJCheckBoxMenuItem getRJCheckBoxMenuItem(java
.lang.String textvar0, boolean bvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJCheckBoxMenuItem getRJCheckBoxMenuItem(java
.lang.String textvar0, javax.swing.Icon iconvar1, boolean bvar2) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJColorChooser getRJColorChooser() throws Rem
oteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RJColorChooser getRJColorChooser(java.awt.Col
or initialColorvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJColorChooser getRJColorChooser(javax.swing.
colorchooser.ColorSelectionModel modelvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJComboBox getRJComboBox(javax.swing.ComboBox
Model aModelvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJComboBox getRJComboBox(java.lang.Object[] i
temsvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJComboBox getRJComboBox(java.util.Vector ite
msvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJComboBox getRJComboBox() throws RemoteExcep
tion ;
        public edu.columbia.cs.cgui.rjfc.swing.RJDesktopPane getRJDesktopPane() throws Remot
eException ;
        public edu.columbia.cs.cgui.rjfc.swing.RJDialog getRJDialog() throws RemoteException
 ;
        public edu.columbia.cs.cgui.rjfc.swing.RJDialog getRJDialog(edu.columbia.cs.cgui.rjf
c.awt.RFrame ownervar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJDialog getRJDialog(edu.columbia.cs.cgui.rjf
c.awt.RFrame ownervar0, boolean modalvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJDialog getRJDialog(edu.columbia.cs.cgui.rjf
c.awt.RFrame ownervar0, java.lang.String titlevar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJDialog getRJDialog(edu.columbia.cs.cgui.rjf
c.awt.RFrame ownervar0, java.lang.String titlevar1, boolean modalvar2) throws RemoteExceptio
n;
        public edu.columbia.cs.cgui.rjfc.swing.RJDialog getRJDialog(edu.columbia.cs.cgui.rjf
c.awt.RDialog ownervar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJDialog getRJDialog(edu.columbia.cs.cgui.rjf
c.awt.RDialog ownervar0, boolean modalvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJDialog getRJDialog(edu.columbia.cs.cgui.rjf
c.awt.RDialog ownervar0, java.lang.String titlevar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJDialog getRJDialog(edu.columbia.cs.cgui.rjf
c.awt.RDialog ownervar0, java.lang.String titlevar1, boolean modalvar2) throws RemoteExcepti
on;
        public edu.columbia.cs.cgui.rjfc.swing.RJEditorPane getRJEditorPane() throws RemoteE
xception ;
        public edu.columbia.cs.cgui.rjfc.swing.RJEditorPane getRJEditorPane(java.net.URL ini
tialPagevar0) throws RemoteException, java.io.IOException;
        public edu.columbia.cs.cgui.rjfc.swing.RJEditorPane getRJEditorPane(java.lang.String
 urlvar0) throws RemoteException, java.io.IOException;
        public edu.columbia.cs.cgui.rjfc.swing.RJEditorPane getRJEditorPane(java.lang.String
 typevar0, java.lang.String textvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJFileChooser getRJFileChooser() throws Remot
eException ;
        public edu.columbia.cs.cgui.rjfc.swing.RJFileChooser getRJFileChooser(java.lang.Stri
ng currentDirectoryPathvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJFileChooser getRJFileChooser(java.io.File c
urrentDirectoryvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJFileChooser getRJFileChooser(edu.columbia.c
s.cgui.rjfc.swing.filechooser.RFileSystemView fsvvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJFileChooser getRJFileChooser(java.io.File c
urrentDirectoryvar0, edu.columbia.cs.cgui.rjfc.swing.filechooser.RFileSystemView fsvvar1) th
rows RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJFileChooser getRJFileChooser(java.lang.Stri
ng currentDirectoryPathvar0, edu.columbia.cs.cgui.rjfc.swing.filechooser.RFileSystemView fsv
var1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJFrame getRJFrame() throws RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RJFrame getRJFrame(java.awt.GraphicsConfigura
```

```
tion gcvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJFrame getRJFrame(java.lang.String titlevar0
) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJFrame getRJFrame(java.lang.String titlevar0
, java.awt.GraphicsConfiguration gcvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJInternalFrame getRJInternalFrame() throws R
emoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RJInternalFrame getRJInternalFrame(java.lang.
String titlevar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJInternalFrame getRJInternalFrame(java.lang.
String titlevar0, boolean resizablevar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJInternalFrame getRJInternalFrame(java.lang.
String titlevar0, boolean resizablevar1, boolean closablevar2) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJInternalFrame getRJInternalFrame(java.lang.
String titlevar0, boolean resizablevar1, boolean closablevar2, boolean maximizablevar3) thro
ws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJInternalFrame getRJInternalFrame(java.lang.
String titlevar0, boolean resizablevar1, boolean closablevar2, boolean maximizablevar3, bool
ean iconifiablevar4) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJLabel getRJLabel(java.lang.String textvar0,
 javax.swing.Icon iconvar1, int horizontalAlignmentvar2) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJLabel getRJLabel(java.lang.String textvar0,
 int horizontalAlignmentvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJLabel getRJLabel(java.lang.String textvar0)
 throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJLabel getRJLabel(javax.swing.Icon imagevar0
, int horizontalAlignmentvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJLabel getRJLabel(javax.swing.Icon imagevar0
) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJLabel getRJLabel() throws RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RJLayeredPane getRJLayeredPane() throws Remot
eException ;
        public edu.columbia.cs.cgui.rjfc.swing.RJList getRJList(javax.swing.ListModel dataMo
delvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJList getRJList(java.lang.Object[] listDatav
ar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJList getRJList(java.util.Vector listDatavar
0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJList getRJList() throws RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RJMenuBar getRJMenuBar() throws RemoteExcepti
on ;
        public edu.columbia.cs.cgui.rjfc.swing.RJMenuItem getRJMenuItem() throws RemoteExcep
tion ;
        public edu.columbia.cs.cgui.rjfc.swing.RJMenuItem getRJMenuItem(javax.swing.Icon ico
nvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJMenuItem getRJMenuItem(java.lang.String tex
tvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJMenuItem getRJMenuItem(javax.swing.Action a
var0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJMenuItem getRJMenuItem(java.lang.String tex
tvar0, javax.swing.Icon iconvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJMenuItem getRJMenuItem(java.lang.String tex
tvar0, int mnemonicvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJMenu getRJMenu() throws RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RJMenu getRJMenu(java.lang.String svar0) thro
ws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJMenu getRJMenu(javax.swing.Action avar0) th
rows RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJMenu getRJMenu(java.lang.String svar0, bool
ean bvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJOptionPane getRJOptionPane() throws RemoteE
xception ;
        public edu.columbia.cs.cgui.rjfc.swing.RJOptionPane getRJOptionPane(java.lang.Object
 messagevar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJOptionPane getRJOptionPane(java.lang.Object
```

```
messagevar0, int messageTypevar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJOptionPane getRJOptionPane(java.lang.Object
messagevar0, int messageTypevar1, int optionTypevar2) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJOptionPane getRJOptionPane(java.lang.Object
messagevar0, int messageTypevar1, int optionTypevar2, javax.swing.Icon iconvar3) throws Rem
oteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJOptionPane getRJOptionPane(java.lang.Object
messagevar0, int messageTypevar1, int optionTypevar2, javax.swing.Icon iconvar3, java.lang.
Object[] optionsvar4) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJOptionPane getRJOptionPane(java.lang.Object
messagevar0, int messageTypevar1, int optionTypevar2, javax.swing.Icon iconvar3, java.lang.
Object[] optionsvar4, java.lang.Object initialValuevar5) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJPanel getRJPanel(java.awt.LayoutManager lay
outvar0, boolean isDoubleBufferedvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJPanel getRJPanel(java.awt.LayoutManager lay
outvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJPanel getRJPanel(boolean isDoubleBufferedva
r0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJPanel getRJPanel() throws RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RJPasswordField getRJPasswordField() throws R
emoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RJPasswordField getRJPasswordField(java.lang.
String textvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJPasswordField getRJPasswordField(int column
svar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJPasswordField getRJPasswordField(java.lang.
String textvar0, int columnsvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJPasswordField getRJPasswordField(javax.swin
g.text.Document docvar0, java.lang.String txtvar1, int columnsvar2) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJPopupMenu getRJPopupMenu() throws RemoteExc
eption ;
        public edu.columbia.cs.cgui.rjfc.swing.RJPopupMenu getRJPopupMenu(java.lang.String l
abelvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJProgressBar getRJProgressBar() throws Remot
eException ;
        public edu.columbia.cs.cgui.rjfc.swing.RJProgressBar getRJProgressBar(int orientvar0
) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJProgressBar getRJProgressBar(int minvar0, i
nt maxvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJProgressBar getRJProgressBar(int orientvar0
, int minvar1, int maxvar2) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJProgressBar getRJProgressBar(javax.swing.Bo
undedRangeModel newModelvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJRadioButton getRJRadioButton() throws Remot
eException ;
        public edu.columbia.cs.cgui.rjfc.swing.RJRadioButton getRJRadioButton(javax.swing.Ic
on iconvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJRadioButton getRJRadioButton(javax.swing.Ac
tion avar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJRadioButton getRJRadioButton(javax.swing.Ic
on iconvar0, boolean selectedvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJRadioButton getRJRadioButton(java.lang.Stri
ng textvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJRadioButton getRJRadioButton(java.lang.Stri
ng textvar0, boolean selectedvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJRadioButton getRJRadioButton(java.lang.Stri
ng textvar0, javax.swing.Icon iconvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJRadioButton getRJRadioButton(java.lang.Stri
ng textvar0, javax.swing.Icon iconvar1, boolean selectedvar2) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJRadioButtonMenuItem getRJRadioButtonMenuIte
m() throws RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RJRadioButtonMenuItem getRJRadioButtonMenuIte
m(javax.swing.Icon iconvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJRadioButtonMenuItem getRJRadioButtonMenuIte
m(java.lang.String textvar0) throws RemoteException;
```

```
        public edu.columbia.cs.cgui.rjfc.swing.RJRadioButtonMenuItem getRJRadioButtonMenuIte
m(javax.swing.Action avar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJRadioButtonMenuItem getRJRadioButtonMenuIte
m(java.lang.String textvar0, javax.swing.Icon iconvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJRadioButtonMenuItem getRJRadioButtonMenuIte
m(java.lang.String textvar0, boolean selectedvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJRadioButtonMenuItem getRJRadioButtonMenuIte
m(javax.swing.Icon iconvar0, boolean selectedvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJRadioButtonMenuItem getRJRadioButtonMenuIte
m(java.lang.String textvar0, javax.swing.Icon iconvar1, boolean selectedvar2) throws RemoteE
xception;
        public edu.columbia.cs.cgui.rjfc.swing.RJRootPane getRJRootPane() throws RemoteExcep
tion ;
        public edu.columbia.cs.cgui.rjfc.swing.RJScrollBar getRJScrollBar(int orientationvar
0, int valuevar1, int extentvar2, int minvar3, int maxvar4) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJScrollBar getRJScrollBar(int orientationvar
0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJScrollBar getRJScrollBar() throws RemoteExc
eption ;
        public edu.columbia.cs.cgui.rjfc.swing.RJScrollPane getRJScrollPane(edu.columbia.cs.
cgui.rjfc.awt.RComponent viewvar0, int vsbPolicyvar1, int hsbPolicyvar2) throws RemoteExcept
ion;
        public edu.columbia.cs.cgui.rjfc.swing.RJScrollPane getRJScrollPane(edu.columbia.cs.
cgui.rjfc.awt.RComponent viewvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJScrollPane getRJScrollPane(int vsbPolicyvar
0, int hsbPolicyvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJScrollPane getRJScrollPane() throws RemoteE
xception ;
        public edu.columbia.cs.cgui.rjfc.swing.RJSeparator getRJSeparator() throws RemoteExc
eption ;
        public edu.columbia.cs.cgui.rjfc.swing.RJSeparator getRJSeparator(int orientationvar
0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJSlider getRJSlider() throws RemoteException
 ;
        public edu.columbia.cs.cgui.rjfc.swing.RJSlider getRJSlider(int orientationvar0) thr
ows RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJSlider getRJSlider(int minvar0, int maxvar1
) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJSlider getRJSlider(int minvar0, int maxvar1
, int valuevar2) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJSlider getRJSlider(int orientationvar0, int
 minvar1, int maxvar2, int valuevar3) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJSlider getRJSlider(javax.swing.BoundedRange
Model brmvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJSplitPane getRJSplitPane() throws RemoteExc
eption ;
        public edu.columbia.cs.cgui.rjfc.swing.RJSplitPane getRJSplitPane(int newOrientation
var0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJSplitPane getRJSplitPane(int newOrientation
var0, boolean newContinuousLayoutvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJSplitPane getRJSplitPane(int newOrientation
var0, edu.columbia.cs.cgui.rjfc.awt.RComponent newLeftComponentvar1, edu.columbia.cs.cgui.rj
fc.awt.RComponent newRightComponentvar2) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJSplitPane getRJSplitPane(int newOrientation
var0, boolean newContinuousLayoutvar1, edu.columbia.cs.cgui.rjfc.awt.RComponent newLeftCompo
nentvar2, edu.columbia.cs.cgui.rjfc.awt.RComponent newRightComponentvar3) throws RemoteExcep
tion;
        public edu.columbia.cs.cgui.rjfc.swing.RJTabbedPane getRJTabbedPane() throws RemoteE
xception ;
        public edu.columbia.cs.cgui.rjfc.swing.RJTabbedPane getRJTabbedPane(int tabPlacement
var0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJTable getRJTable() throws RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RJTable getRJTable(javax.swing.table.TableMod
el dmvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJTable getRJTable(javax.swing.table.TableMod
```

```
el dmvar0, javax.swing.table.TableColumnModel cmvar1) throws R    teException;
    ,     public edu.columbia.cs.cgui.rjfc.swing.RJTable getRJTable(javax.swing.table.TableMod
el dmvar0, javax.swing.table.TableColumnModel cmvar1, javax.swing.ListSelectionModel smvar2)
 throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJTable getRJTable(int numRowsvar0, int numCo
lumnsvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJTable getRJTable(java.util.Vector rowDatava
r0, java.util.Vector columnNamesvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJTable getRJTable(java.lang.Object[][] rowDa
tavar0, java.lang.Object[] columnNamesvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJTextArea getRJTextArea() throws RemoteExcep
tion ;
        public edu.columbia.cs.cgui.rjfc.swing.RJTextArea getRJTextArea(java.lang.String tex
tvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJTextArea getRJTextArea(int rowsvar0, int co
lumnsvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJTextArea getRJTextArea(java.lang.String tex
tvar0, int rowsvar1, int columnsvar2) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJTextArea getRJTextArea(javax.swing.text.Doc
ument docvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJTextArea getRJTextArea(javax.swing.text.Doc
ument docvar0, java.lang.String textvar1, int rowsvar2, int columnsvar3) throws RemoteExcept
ion;
        public edu.columbia.cs.cgui.rjfc.swing.RJTextField getRJTextField() throws RemoteExc
eption ;
        public edu.columbia.cs.cgui.rjfc.swing.RJTextField getRJTextField(java.lang.String t
extvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJTextField getRJTextField(int columnsvar0) t
hrows RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJTextField getRJTextField(java.lang.String t
extvar0, int columnsvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJTextField getRJTextField(javax.swing.text.D
ocument docvar0, java.lang.String textvar1, int columnsvar2) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJTextPane getRJTextPane() throws RemoteExcep
tion ;
        public edu.columbia.cs.cgui.rjfc.swing.RJTextPane getRJTextPane(javax.swing.text.Sty
ledDocument docvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJToggleButton getRJToggleButton() throws Rem
oteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RJToggleButton getRJToggleButton(javax.swing.
Icon iconvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJToggleButton getRJToggleButton(javax.swing.
Icon iconvar0, boolean selectedvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJToggleButton getRJToggleButton(java.lang.St
ring textvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJToggleButton getRJToggleButton(java.lang.St
ring textvar0, boolean selectedvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJToggleButton getRJToggleButton(javax.swing.
Action avar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJToggleButton getRJToggleButton(java.lang.St
ring textvar0, javax.swing.Icon iconvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJToggleButton getRJToggleButton(java.lang.St
ring textvar0, javax.swing.Icon iconvar1, boolean selectedvar2) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJToolBar getRJToolBar() throws RemoteExcepti
on ;
        public edu.columbia.cs.cgui.rjfc.swing.RJToolBar getRJToolBar(int orientationvar0) t
hrows RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJToolBar getRJToolBar(java.lang.String namev
ar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJToolBar getRJToolBar(java.lang.String namev
ar0, int orientationvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJToolTip getRJToolTip() throws RemoteExcepti
on ;
        public edu.columbia.cs.cgui.rjfc.swing.RJTree getRJTree() throws RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RJTree getRJTree(java.lang.Object[] valuevar0
```

```
) throws RemoteException;
    '     public edu.columbia.cs.cgui.rjfc.swing.RJTree getRJTree(java.util.Vector valuevar0)
throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJTree getRJTree(java.util.Hashtable valuevar
0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJTree getRJTree(javax.swing.tree.TreeNode ro
otvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJTree getRJTree(javax.swing.tree.TreeNode ro
otvar0, boolean asksAllowsChildrenvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJTree getRJTree(javax.swing.tree.TreeModel n
ewModelvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJViewport getRJViewport() throws RemoteExcep
tion ;
        public edu.columbia.cs.cgui.rjfc.swing.RJWindow getRJWindow() throws RemoteException
 ;
        public edu.columbia.cs.cgui.rjfc.swing.RJWindow getRJWindow(java.awt.GraphicsConfigu
ration gcvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJWindow getRJWindow(edu.columbia.cs.cgui.rjf
c.awt.RFrame ownervar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJWindow getRJWindow(edu.columbia.cs.cgui.rjf
c.awt.RWindow ownervar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RJWindow getRJWindow(edu.columbia.cs.cgui.rjf
c.awt.RWindow ownervar0, java.awt.GraphicsConfiguration gcvar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RMenuSelectionManager getRMenuSelectionManage
r() throws RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.ROverlayLayout getROverlayLayout(edu.columbia
.cs.cgui.rjfc.awt.RContainer targetvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RProgressMonitorInputStream getRProgressMonit
orInputStream(edu.columbia.cs.cgui.rjfc.awt.RComponent parentComponentvar0, java.lang.Object
 messagevar1, java.io.InputStream invar2) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RProgressMonitor getRProgressMonitor(edu.colu
mbia.cs.cgui.rjfc.awt.RComponent parentComponentvar0, java.lang.Object messagevar1, java.lan
g.String notevar2, int minvar3, int maxvar4) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RRepaintManager getRRepaintManager() throws R
emoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RScrollPaneLayout getRScrollPaneLayout() thro
ws RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RSizeRequirements getRSizeRequirements() thro
ws RemoteException ;
        public edu.columbia.cs.cgui.rjfc.swing.RSizeRequirements getRSizeRequirements(int mi
nvar0, int prefvar1, int maxvar2, float avar3) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RSizeSequence getRSizeSequence() throws Remot
eException ;
        public edu.columbia.cs.cgui.rjfc.swing.RSizeSequence getRSizeSequence(int numEntries
var0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RSizeSequence getRSizeSequence(int numEntries
var0, int valuevar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RSizeSequence getRSizeSequence(int[] sizesvar
0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RTimer getRTimer(int delayvar0, edu.columbia.
cs.cgui.rjfc.event.RActionListener listenervar1) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RUIDefaults getRUIDefaults() throws RemoteExc
eption ;
        public edu.columbia.cs.cgui.rjfc.swing.RUIDefaults getRUIDefaults(java.lang.Object[]
 keyValueListvar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RUIManager getRUIManager() throws RemoteExcep
tion ;
        public edu.columbia.cs.cgui.rjfc.swing.RUnsupportedLookAndFeelException getRUnsuppor
tedLookAndFeelException(java.lang.String svar0) throws RemoteException;
        public edu.columbia.cs.cgui.rjfc.swing.RViewportLayout getRViewportLayout() throws R
emoteException ;

}
```

CLAIMS

1. A method for distributed processing through a server and a remote client wherein an application is executed entirely in the server, wherein the application is configured to interact with a user interface toolkit according to an application programming interface, and wherein the user interface toolkit has a component that performs a function, the method comprising:

providing the user interface toolkit on the remote client such that the component is configured to perform the function on the remote client;

providing a remote-capable user interface toolkit on the server by creating a remote-capable component which is configured to interact with the application according to the application programming interface and which is configured to generate a message to the component on the remote client to perform the respective function on the remote client;

invoking the remote-capable user interface toolkit by the application to perform a function according to the application programming interface ;

generating the message to perform the function by the remote-capable component of the remote-capable user interface toolkit on the server in response to the invocation by the application;

communicating the message between the remote-capable user interface toolkit on the server and the user interface toolkit on the remote client; and

performing the function on the remote client by the component of the user interface toolkit in response to the message.

2. The method of claim 1, wherein the component in the user interface toolkit is configured to render a graphical item and the remote-capable component is configured to generate a message to render the graphical item, and wherein communicating the message between the remote-capable user interface toolkit on the server and the user interface toolkit on the remote client comprises transmitting the message to the user interface toolkit on the remote client to render the graphical item.

3. The method of claim 2, wherein performing the function on the remote client by the component of the user interface toolkit comprises rendering the graphical item on the remote client in response to the message.

4. The method of claim 1, wherein the component in the user interface toolkit is configured to install an event handler and the remote-capable component is configured to generate a message to install the event handler, and wherein communicating the message between the remote-capable user interface toolkit on the server and the user interface toolkit on the remote client comprises transmitting the message to the user interface toolkit on the remote client to install an event handler.

5. The method of claim 4, wherein performing the function on the remote client by the component of the user interface toolkit comprises installing the event handler on the remote client in response to the message.

6. The method of claim 1, which further comprises:

generating an event by the remote-capable component of the remote-capable user interface toolkit in response to the step of invoking; and

wherein communicating the message between the remote-capable user interface toolkit on the server and the user interface toolkit on the remote client comprises asynchronously transmitting the event to the user interface toolkit.

7. The method of claim 6, wherein the application is a database searching application configured to search a database for information in response to a user-defined request,

wherein the step of generating an event by the remote-capable component of the remote-capable user interface toolkit comprises identifying information from the database in response to the user-defined request; and

wherein the step of asynchronously transmitting the event to the user interface toolkit comprises asynchronously transmitting a message to the remote client to render the information from the database identified in the step of generating an event.

8. The method of claim 7, wherein the application is a web browser and wherein the database is the World Wide Web,

wherein the step of identifying information from the database comprises identifying information from the World Wide Web; and

wherein the step of asynchronously transmitting a command to the remote client to render the information from the database comprises asynchronously transmitting a command to the remote client to render the information from the World Wide Web.

9. The method of claim 1 wherein the step of providing a remote-capable user interface toolkit on the server further comprises:

providing a code-generating computer program configured to read in the code of the component of the user interface toolkit and to generate the remote-capable component of the remote-capable user interface toolkit by substituting a portion of the code relevant to executing the function with a portion of code configured to issue a remote command to execute the function;

reading in the code of the component of the user interface toolkit;

generating the remote-capable component of the remote-capable user interface toolkit by copying the code of the component and by substituting the portion of the code relevant to executing the function with the portion of code configured to issue the remote command to execute the function

10. A distributed computer system having at least one server and one remote client wherein the server executes the entire application on the server, wherein the application is configured to interact with a user interface toolkit according to an application programming interface, and wherein the user interface toolkit has a component that performs a function, the distributed computer system comprising:

a user interface toolkit on the remote client having a component configured to perform a function on the remote client;

a remote-capable user interface toolkit on the server having a remote-capable component which is configured to interact with the application according to the application programming interface, and which is configured to generate a message to the component on the remote client to perform the respective function on the remote client in response to an invocation of the function by the application;

a server configured to communicate the message between the remote-capable user interface toolkit on the server and the user interface toolkit on the remote client; and

a remote client configured to performing the function by the component of the user interface toolkit in response to the message.

11. The distributed computer system of claim 10, wherein the component in the user interface toolkit is configured to render a graphical item and the remote-capable component is configured to generate a message to render the graphical item

12. The distributed computer system of claim 11, wherein the server is configured to communicate the message to the user interface toolkit on the remote client to render the graphical item.

13. The distributed computer system of claim 12, wherein the component of the user interface toolkit on the remote client is configured to render the graphical item in response to the message.

14. The distributed computer system of claim 10, wherein the component in the user interface toolkit is configured to render an item and the remote-capable component is configured to generate a message to render the item

15. The distributed computer system of claim 14, wherein the server is configured to communicate the message to the user interface toolkit on the remote client to render the item.

16. The distributed computer system of claim 15, wherein the component of the user interface toolkit on the remote client is configured to render the item in response to the message.

17. The distributed computer system of claim 10, wherein the component in the user interface toolkit is configured to install an event handler and the remote-capable component is configured to generate a command to install an event handler.

18. The distributed computer system of claim 17, wherein the server is configured to communicate the message to the user interface toolkit on the remote client to install the event handler.

19. The distributed computer system of claim 18, wherein the component of the user interface toolkit is configured to install the event handler on the remote client in response to the message.

20. The distributed computer system of claim 10:

wherein the remote-capable component of the remote-capable user interface toolkit is configured to generate an event in response to the step of invoking; and

wherein the server is configured to asynchronously communicate a message to generate the event to the user interface toolkit on the remote client.

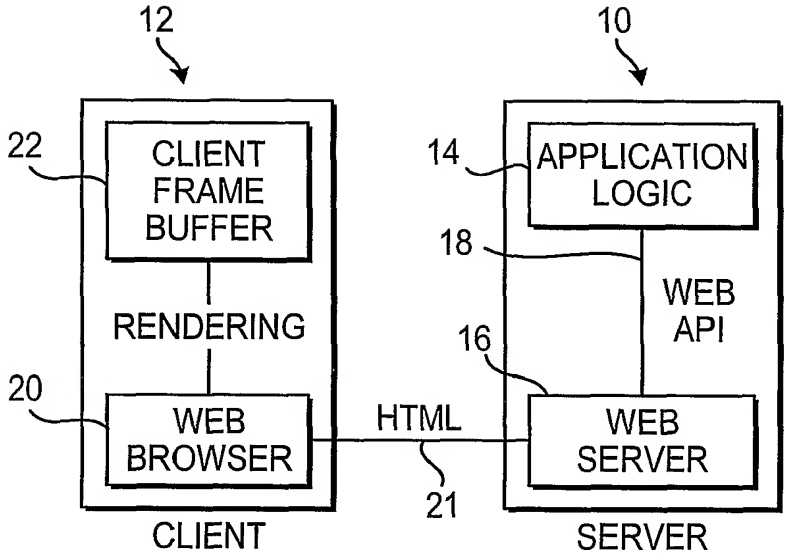


FIG. 1
(PRIOR ART)

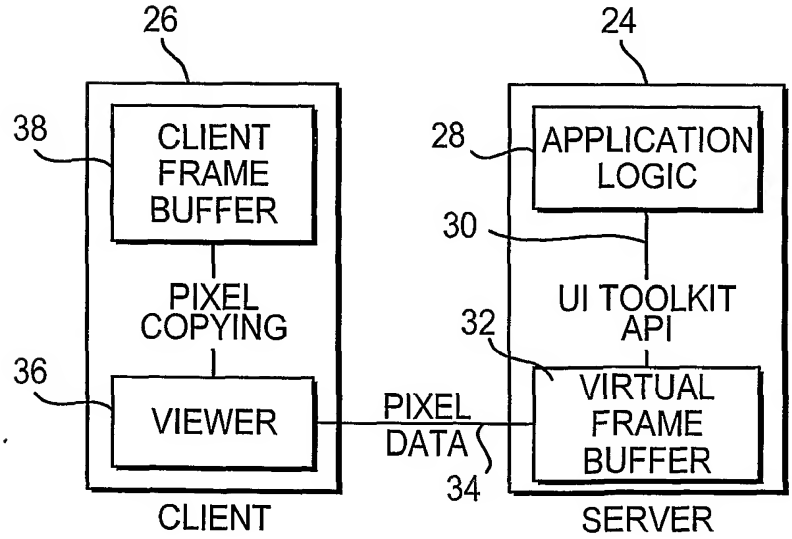


FIG. 2
(PRIOR ART)

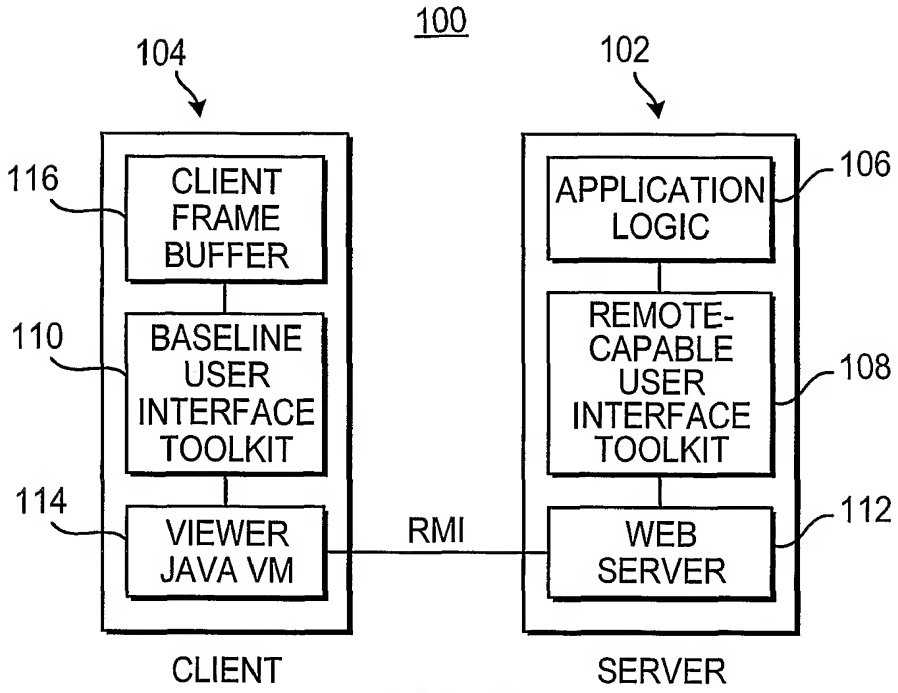


FIG. 3

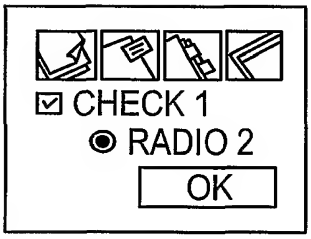


FIG. 4(a)
(PRIOR ART)

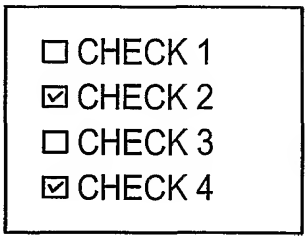


FIG. 4(b)
(PRIOR ART)

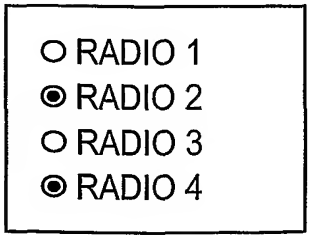


FIG. 4(c)
(PRIOR ART)

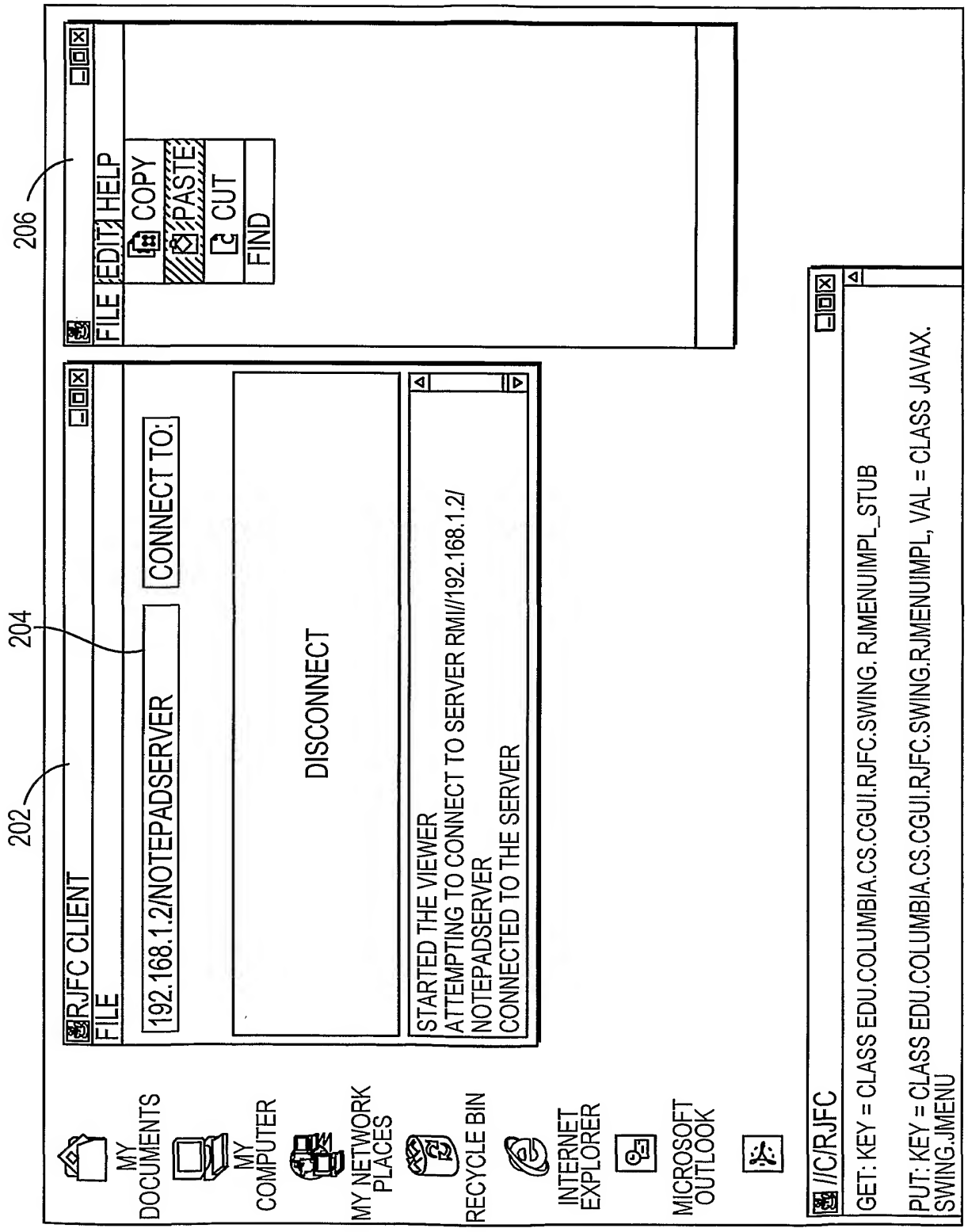

200
202
204
206
RJFC CLIENT
FILE
192.168.1.2/NOTEPADSERVER
CONNECT TO:
DISCONNECT
STARTED THE VIEWER
ATTEMPTING TO CONNECT TO SERVER RMI//192.168.1.2/
NOTEPADSERVER
CONNECTED TO THE SERVER
FILE EDIT HELP
COPY
PASTE
CUT
FIND
MY DOCUMENTS
MY COMPUTER
MY NETWORK PLACES
RECYCLE BIN
INTERNET EXPLORER
MICROSOFT OUTLOOK
//C/RJFC
GET: KEY = CLASS EDU.COLUMBIA.CS.CGUI.RJFC.SWING. RJMENUIMPL_STUB
PUT: KEY = CLASS EDU.COLUMBIA.CS.CGUI.RJFC.SWING.RJMENUIMPL, VAL = CLASS JAVAX.
SWING.JMENU

FIG. 5

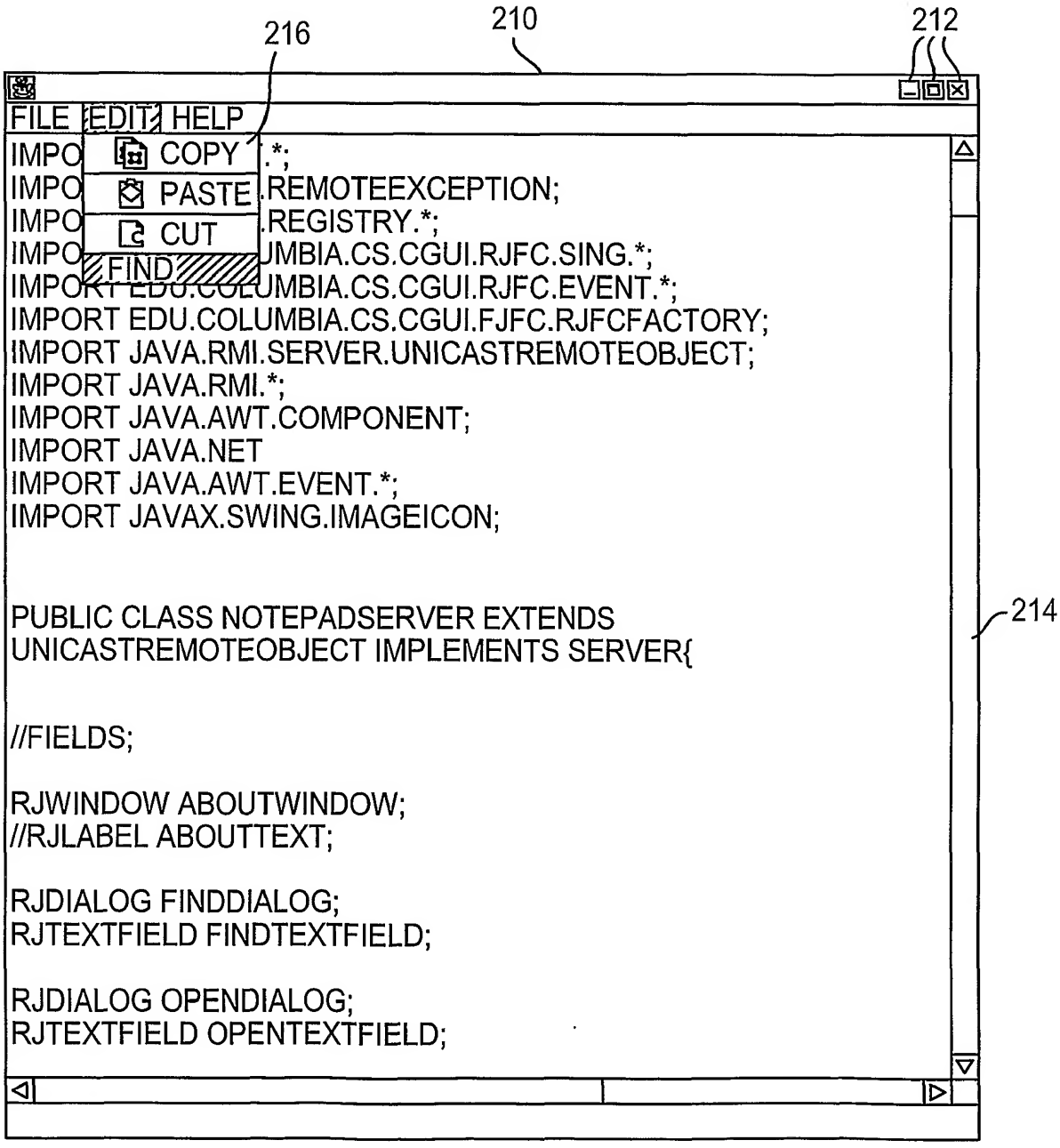

216
210
212
FILE EDIT HELP
COPY
PASTE
CUT
FIND
IMPO .*;
IMPO .REMOTEEXCEPTION;
IMPO .REGISTRY.*;
IMPO UMBIA.CS.CGUI.RJFC.SING.*;
IMPORT EDU.COLUMBIA.CS.CGUI.RJFC.EVENT.*;
IMPORT EDU.COLUMBIA.CS.CGUI.FJFC.RJFCFACTORY;
IMPORT JAVA.RMI.SERVER.UNICASTREMOTEOBJECT;
IMPORT JAVA.RMI.*;
IMPORT JAVA.AWT.COMPONENT;
IMPORT JAVA.NET
IMPORT JAVA.AWT.EVENT.*;
IMPORT JAVAX.SWING.IMAGEICON;
PUBLIC CLASS NOTEPADSERVER EXTENDS
UNICASTREMOTEOBJECT IMPLEMENTS SERVER{
//FIELDS;
RJWINDOW ABOUTWINDOW;
//RJLABEL ABOUTTEXT;
RJDIALOG FINDDIALOG;
RJTEXTFIELD FINDTEXTFIELD;
RJDIALOG OPENDIALOG;
RJTEXTFIELD OPENTEXTFIELD;
214

FIG. 6(a)

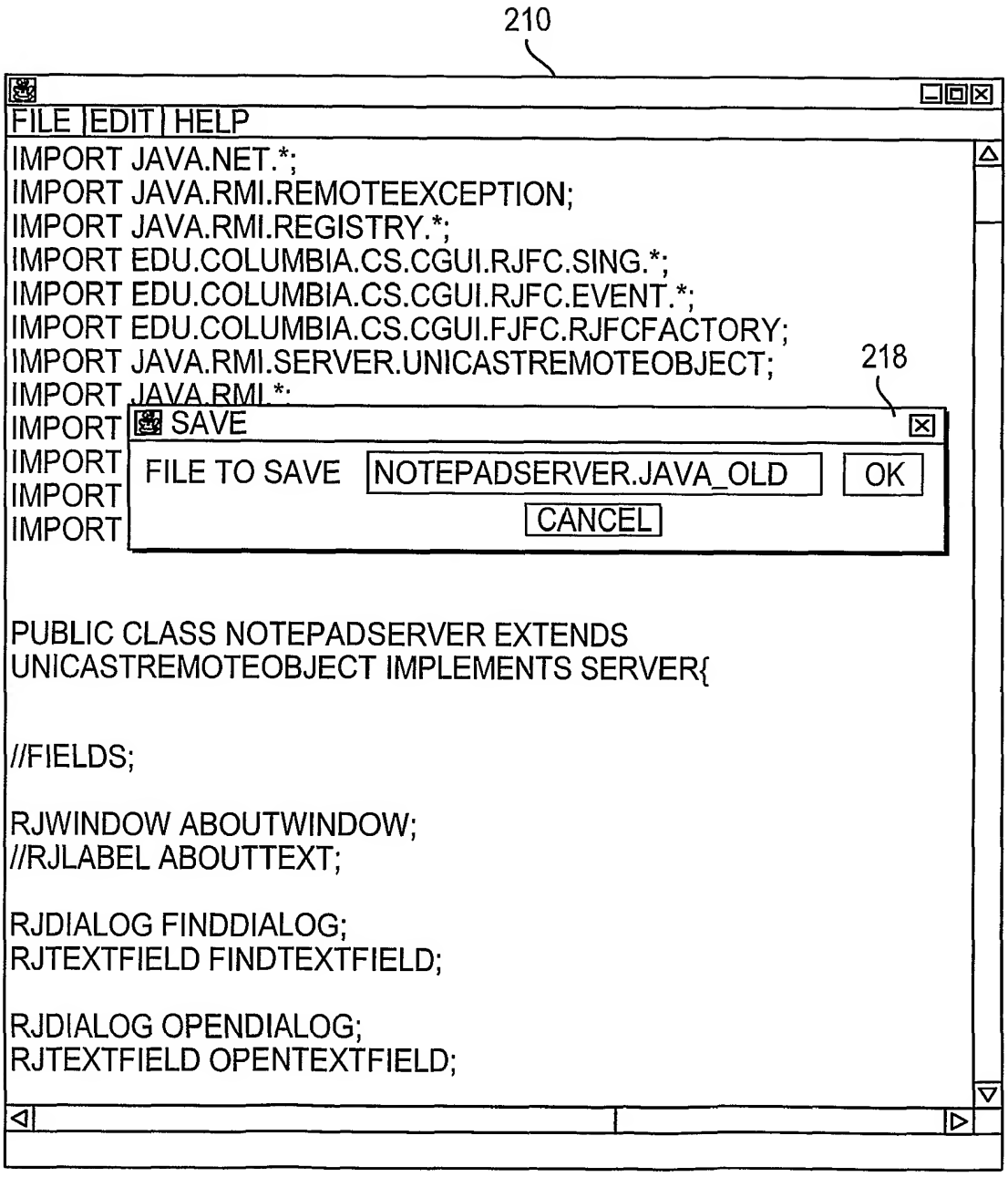

210
FILE EDIT HELP
IMPORT JAVA.NET.*;
IMPORT JAVA.RMI.REMOTEEXCEPTION;
IMPORT JAVA.RMI.REGISTRY.*;
IMPORT EDU.COLUMBIA.CS.CGUI.RJFC.SING.*;
IMPORT EDU.COLUMBIA.CS.CGUI.RJFC.EVENT.*;
IMPORT EDU.COLUMBIA.CS.CGUI.FJFC.RJFCFACTORY;
IMPORT JAVA.RMI.SERVER.UNICASTREMOTEOBJECT;
IMPORT JAVA.RMI.*;
218
SAVE
FILE TO SAVE
NOTEPADSERVER.JAVA_OLD
OK
CANCEL
PUBLIC CLASS NOTEPADSERVER EXTENDS
UNICASTREMOTEOBJECT IMPLEMENTS SERVER{
//FIELDS;
RJWINDOW ABOUTWINDOW;
//RJLABEL ABOUTTEXT;
RJDIALOG FINDDIALOG;
RJTEXTFIELD FINDTEXTFIELD;
RJDIALOG OPENDIALOG;
RJTEXTFIELD OPENTEXTFIELD;

FIG. 6(b)

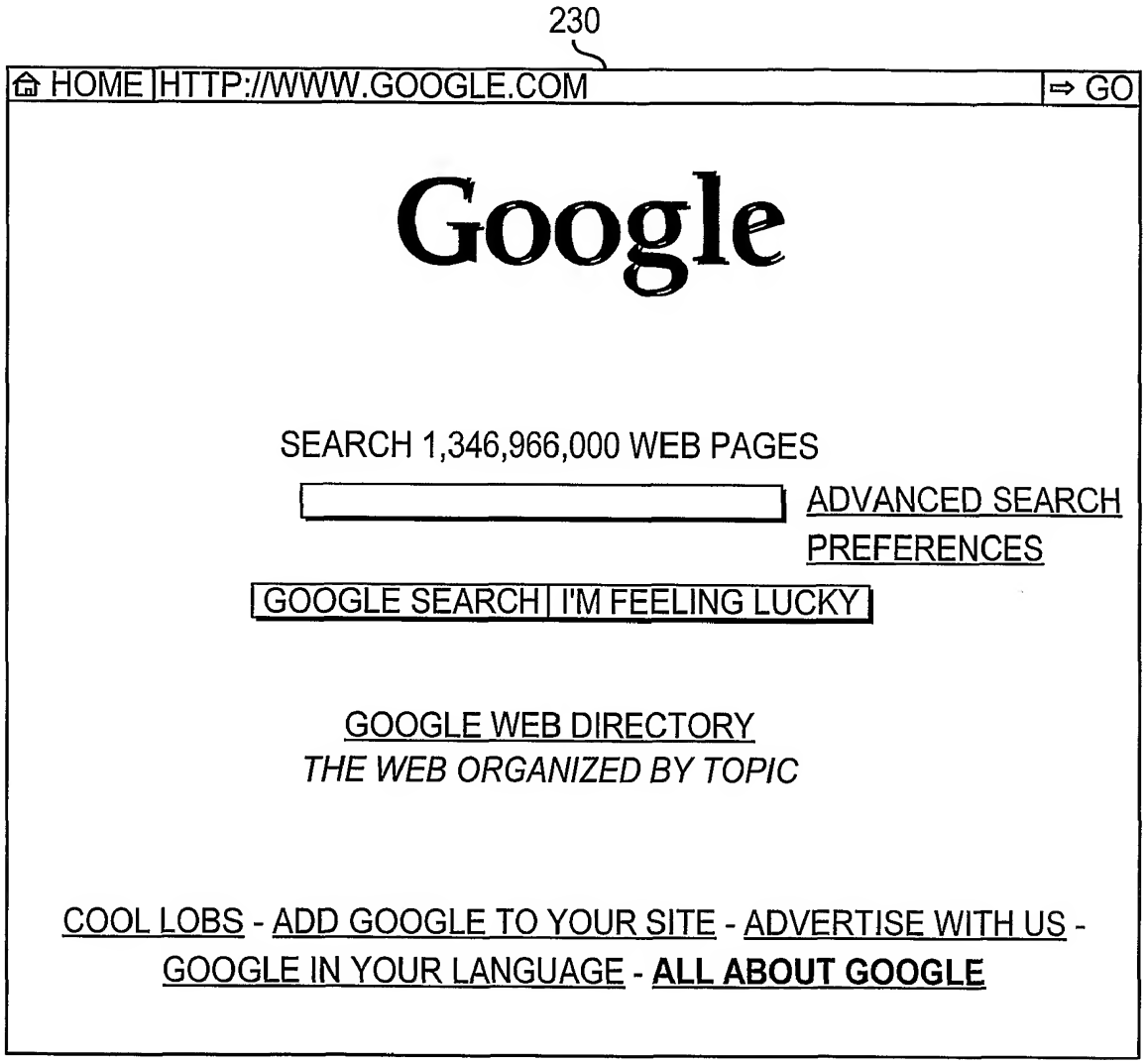

230
HOME
HTTP://WWW.GOOGLE.COM
GO
Google
SEARCH 1,346,966,000 WEB PAGES
ADVANCED SEARCH
PREFERENCES
GOOGLE SEARCH
I'M FEELING LUCKY
GOOGLE WEB DIRECTORY
THE WEB ORGANIZED BY TOPIC
COOL LOBS - ADD GOOGLE TO YOUR SITE - ADVERTISE WITH US -
GOOGLE IN YOUR LANGUAGE - ALL ABOUT GOOGLE

FIG. 7

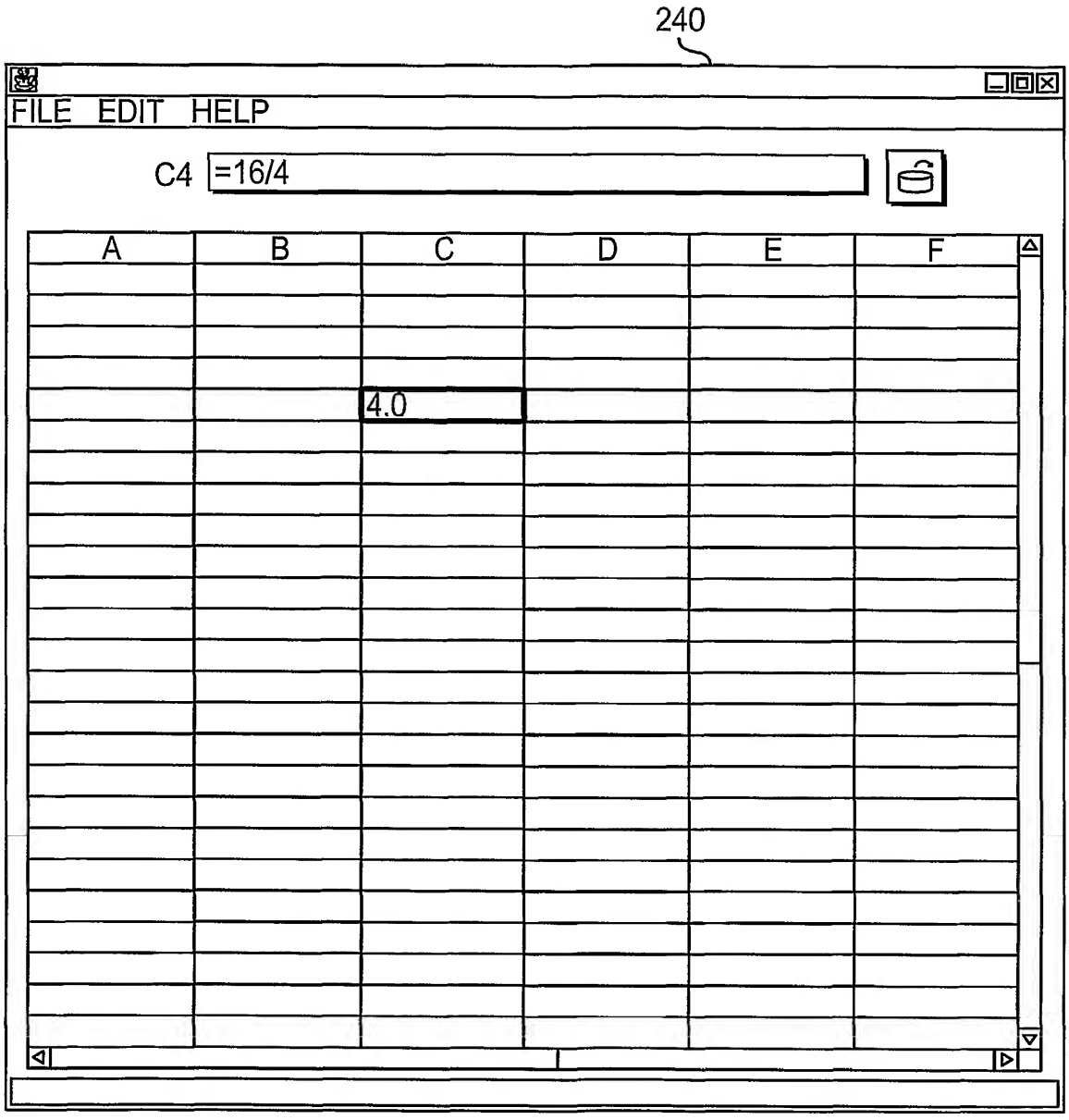


FIG. 8

```
PUBLIC VOID REGISTERDISPLAY (RJFRAME D,
        RJFCFACTORY F) THROWS REMOTEEXCEPTION {

    RJTEXTAREA THEAREA = F.GETRJTEXTAREA (20,20) ;
    THEAREA.ADDKEYLISTENER (NEW)
                        TEXTAREAKEYLISTENER( ) ) ;

    RJSCROLLPANE PANE = F.GETRJSCROLLPANE( ) ;
    PANE.SETVIEWPORTVIEW(TEXTAREA) ;

    RJTEXTFIELD STATUSBAR = F.GETRJTEXTFIELD( ) ;
    STATUSBAR.SETEDITABLE (FALSE) ;

    RCONTAINER RC = D.GETCONTENTPANE ( ) ;
    RC.SETLAYOUT (NEW BORDERLAYOUT ( ( ) ;
    RC.ADD (STATUSBAR, BORDERLAYOUT.SOUTH) ;
    RC.ADD (CREATEMENU ( ) , BORDERLAYOUT.NORTH) ;
    RC.ADD (PANE, BORDERLAYOUT.CENTER) ;

}
```

FIG. 9(a)

```
PUBLIC MYJFRAME ( ) EXTENDS JFRAME {

    JTEXTAREA THEAREA = NEW JTEXTAREA (20,20) ;
    THEAREA.ADDKEYLISTENER (NEW)
                        TEXTAREAKEYLISTENER( ) ) ;

    JSCROLLPANE PANE = NEW JSCROLLPANE( ) ;
    PANE.SETVIEWPORTVIEW(TEXTAREA) ;
    JTEXTFIELD STATUSBAR = NEW JTEXTFIELD( )
    ;
    STATUSBAR.SETEDITABLE (FALSE) ;

    CONTAINER C = THIS.GETCONTENTPANE ( ) ;
    C.SETLAYOUT (NEW BORDERLAYOUT ( ( ) ;
    C.ADD (STATUSBAR, BORDERLAYOUT.SOUTH) ;
    C.ADD (CREATEMENU ( ) , BORDERLAYOUT.NORTH) ;
    C.ADD (PANE, BORDERLAYOUT.CENTER) ;

}
```

FIG. 9(b)
(PRIOR ART)